Multi-Type Nearest and Reverse Nearest Neighbor Search : Concepts and Algorithms

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Xiaobin Ma

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Shashi Shekhar

Name of Faculty Adviser(s)

February 2012

Acknowledgments

This work represents the culmination of many years of work, and would never have been completed without the support and contributions of many people. First and foremost I would like to thank my academic adviser, Professor Shashi Shekhar for invaluable guidance, enormous patience and unwavering support in all respects. Without his patient support it is not possible to make this dissertation a reality. He not only helped provide the insight, patience, and knowledge necessary to accomplish successfully but also taught me the approaches to do research, write paper and give presentations.

I would like to thank all members of my committee, Professor Jaideep Srivastava, Professor Mohamed Mokbel and Professor Gediminas Adomavicius, for their time in reviewing my work and giving suggestive advices. I would also thank Professor Hui Xiong, Professor Yan Huang, Dr. Chengyang Zhang and Pusheng Zhang for their discussions about the work and reviewing the papers.

I owe more thanks and love than anyone could possibly imagine to the people - my parents, Kanchu Ma and Zhifang Wang, my grand parents Mingshan Wang and Changli Bai, and my parent-in-law Renkui Lu. They have been encouraging me during the studies as they always do in my life.

My deepest thanks and love to my wife Jiping, and my son Kevin and daughter Lillian. They bring the warmness and happiness to my life and work, which solidly support my long time study.

I would like specially thank Kimberly Koffolt, who helped to make me a better writer and presenter. She always patiently reads and polishes every paper I wrote. I also owe too many thanks to my colleagues from spatial database group in University of Minnesota and friends Betsy George, Pradeep Mohan, Mike Evans, Yu Liang, Xiuzhen Cheng, Dechang Chen, Weili Wu, Chang-Tien Lu, Xinping Zhang, Zhihong

Yao, Yu Ming and Ningsheng Huang. Without their friendship and help, this work would not have been accomplished.

Finally, I would like to thank the University of Minnesota Computer Science & Engineering for providing me the best equipment and facilities. Special thanks to the system staff who establish and maintain the very efficient computing environment.

Dedication

To my parents, Jiping, Kevin and Lillian

Abstract

The growing availability of spatial databases and the computational resources to exploit them has led to Geographic Information Systems (GIS) of increasing complexity. At the same time, users' expectations of location-based services are also growing, which means these services must be able to handle ever more complex queries. This thesis investigates methods that expand the scope of traditional database search methods for answering location-based queries in today's computing environment. Traditionally, location-based services and other applications have relied heavily on two concepts: the nearest neighbor search (known also as the proximity, similarity, or closest point search) and the reverse nearest neighbor search.

Given a query point q and a set S of points in metric space, the nearest neighbor (NN) search finds a point p in S such that the distance from q to p is shortest. An example NN query might be "where is the closest post office to my hotel". Numerous variants of the NN search, including the all-nearest-neighbor, group nearest neighbor and K-closest neighbor search among others, also play a critical role in location-based services.

Related to the classic NN query problem is the Reverse Nearest Neighbor (RNN) query problem, which is normally used to find the "influence" of a point on the database. In applications such as decision support systems, RNN search is widely used to make business decisions. For example, it is used to find the influence of a new supermarket on a neighborhood by finding how many residences have this supermarket as their nearest neighbor.

One fundamental limitation of traditional NN and RNN search queries in today's computing environment, however, is their inability to consider more than one feature type. For example, a NN search can determine the closest post office to a hotel, but not the closest post office, gas station and grocery store, for a traveler who wants the shortest path that starts at a hotel and passes through a post office, a gas station,

and a grocery store. Likewise, classic RNN searches that cannot account for the influence of more than one feature type may be of limited value for decision-makers in competitive business settings.

In this thesis, we attempt to expand the scope of traditional database search methods by exploring the effect of multiple feature types on nearest neighbor and reverse nearest neighbor search queries. We first formally define the notion of the Multi-Type Nearest Neighbor (MTNN) search. Given a query point and a collection of spatial feature types an MTNN query finds the shortest tour for the query point such that one instance of every feature type is visited during the tour. For example, the shortest tour which starts at a hotel and passes through a post office, a gas station, and a grocery store. We propose an R-tree based solution exploiting a page level upper bound for efficient computation in clustered data sets and finding optimal query result, and compare our method to RLORD, an existing method which assumes a fixed order of feature types, by analyzing the cost model and experimenting.

We then research the effect of multiple feature types on real world applications by extending the MTNN search to spatio-temporal road networks. We model the road networks as a time aggregated multi-type graph, a special case of a time aggregated encoded path view. Based on this model, we formalize the BEst Start Time Multi-Type Nearest Neighbor (BESTMTNN) query problem and present new algorithms that give the best start time, a turn-by-turn route and shortest path in terms of least travel time for a given query.

Finally, we study how multiple feature types affect the reverse nearest neighbor search. Traditional RNN searches consider only the effect of the feature type that the query point belongs to. A Multi-Type Reverse Nearest Neighbor (MTRNN) query problem is formalized to capture the notion of finding the influence of a query point and other objects that belong to multiple other feature types in the search space. In other words, the MTRNN query finds all the objects that have the query point and one point from every feature type as their MTNN nearest neighbor. We show that the MTRNN can yield dramatically different results compared to the classic RNN search.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The growing availability of spatial databases and the computational resources to exploit them has led to Geographic Information Systems (GIS) of increasing complexity. Meanwhile, the growing popularity of location-based services is raising users' expectations of these services, which means these services must be able to handle ever more complex queries. This thesis investigates methods that expand the scope of traditional database search methods for answering location-based queries in today's computing environment.

Traditionally, location-based services and other related applications have relied heavily on the concepts of the nearest neighbor search and reverse nearest neighbor search. Given a query point, the problem of the Nearest Neighbor (NN) search in database society [4–6, 10, 12, 23, 29, 42, 45, 49, 51, 52, 57, 61] is to find the closest point in a given data set from a huge database. A traditional NN query can be stated as follows: given a point set $P = \{p_1, p_2, ..., p_n\}$ and a query point q in a vector space, the NN query finds a point p_k such that the distance from q to $p_k \in P$ is minimized among the distances from q to $p_i \in P$. Many application domains make use of the NN query. For example, in Geographic Information Systems (GIS), "find the nearest gas station from my location" is a typical query that uses a NN query technique. The NN problem was first introduced into the spatial database community by Roussopoulos and Kelly in their pioneering paper [49]. Following their work, extensive research were done to tackle classic nearest neighbor search problem and many of the variants such as k-closest pair search [14,15,24,25,52,71], k-nearest neighbor search [16,51,57], all nearest neighbor search [11, 13, 24, 25, 75], group nearest neighbor search [44], and

continuous nearest neighbor search [61]. All of these problem formulations focus on one or two object types and try to find relationships among object points within one or two object types. However, for many application domains, including location-based services, the assumption of only one or two data object types is severely limiting. In many cases, it is the relationship among multiple types of objects that's important. For example, a traveler may want to know not the closest post office to a hotel, but rather the shortest tour that passes through a post office, a gas station, and a grocery store.

Related to the classic NN query problem is the Reverse Nearest Neighbor (RNN) query problem [3, 17, 18, 21, 28, 30–32, 46, 56, 58, 59, 62–66, 68–70, 73]. The RNN search finds all points that have the given query point as their nearest neighbor. This type of search is normally used to find the "influence" of a point on the database. It was first formalized to capture the notion of influence set by Korn and Muthukrishnan in their work [31]. Given a data set P and a query point $f_{q,q}$, an RNN query finds all objects in P that have the query point $f_{q,q}$ as their nearest neighbor. RNN queries have widely been used in Decision Support Systems, Profile-Based Marketing, etc. For example, before deciding to build a new supermarket, a company needs to know how many customers the supermarket may potentially attract. In other words, it needs to know the influence of opening a supermarket. An RNN query can be used to find all residential customers that live closer to the new supermarket than any other supermarket. Following the work [31], an on-line algorithm [58] for dynamic databases and an index structure was devised to answer RNN queries by Yang and Lin in [17]. Similar to NN queries, RNN problems have been studied in an extended family containing different variations, for example, monochromatic and bichromatic RkNN queries [3, 68], visible RkNN problem [21], aggregate RNN over data stream [32], reverse top-k query problem [65], MaxBRNN problem [66], continuous RNN [28, 43, 64, 67–69], and Reverse Skyline Queries [18]. As with the traditional NN problems, however, all the current RNN problems consider the influence of a single feature type. In some applications, this limitation may significantly affect the quality of results and lead to incorrect business decisions.

In this thesis, we address the need for expanded NN and RNN search capabilities in spatial databases and GIS-related applications by studying the effect of multiple feature types on search problems, i.e., the relationship among more than two types of objects. We have formalized the notion of searching the nearest neighbor in objects

of multiple feature types as an MTNN query problem [40]. Given a query point and a collection of spatial features, an MTNN query finds the shortest tour for the query point such that one instance of each feature is visited during the tour. For example, a tourist may be interested in finding the shortest tour which starts at a hotel and passes through a post office, a gas station, and a grocery store. The MTNN query problem differs from the traditional nearest neighbor query problem in that there are potentially many objects for each feature type and the shortest tour should pass through only one object from each feature type. We have studied a generalized MTNN query problem and provided algorithms that find the optimal solution, the shortest route, to the problem. Based on an R-tree index, we have designed an algorithm which exploits a Page-Level Upper Bound (PLUB) for efficient pruning at the R-tree node level. These algorithms are based on a page-level pruning strategy. R-tree page-level pruning method nicely compliments the instance-level pruning method, since it makes better use of the R-tree index for reducing I/O cost. After discussion of our PLUB pruning strategy, we have given a cost model for the PLUB algorithm. Experimental results show that the PLUB algorithm can answer MTNN query within reasonable time.

We then extend our MTNN approach to spatio-temporal road networks using queries exhibiting important spatio-temporal properties [38]. For example, a traveler may be interested in finding a shortest route in terms of least travel time with the best start time between 9:00 am and 11:00 am from his house through one grocery store (with a stay of 1 1/2 hours), one electronics store (1 hour stay) and one post office (arriving before 4:00 pm; 1/2 hour stay) and returning home before 8:00 pm. This query illustrates some important properties. First, the traveler is trying to find a route with instances from different feature types (grocery store, an electric appliances store etc). Second, the route to be found is a closed route from the query point back to the query point. Third, the traveler is interested in not only the route but also the best start time. Considering the variability of traffic patterns at different times on road networks, this best start time could differ for different time windows. Therefore, the query asks for answers containing not only spatial features like the route but also temporal features. Fourth, the query itself contains spatial and temporal features. For example, the query point and different interested locations are spatial features. The best start time between 9:00 am and 11:00 am and length of stay at each location are temporal features. We have formalized this query problem as a BEst Start Time

Multi-Type Nearest Neighbor (BESTMTNN) query problem [38] and proposed a label-correcting based algorithm to solve it. This algorithm prioritizes the spatio-temporal partial routes with current least travel time. It takes a user-specified query that involves spatio-temporal features such as query time window sizes for all features and planned stay time interval at a location and gives a turn-by-turn route and the best start time in terms of least travel time.

Finally we have studied how multiple feature types influence the RNN search and formalized the Multi-Type Reverse Nearest Neighbor (MTRNN) query problem [41]. This work was motivated by the observation that a set of points may be influenced by more than one type of data, not just a single type as is assumed in the classic RNN query. For example, in the query "find all residential customers that live closer to the new supermarket than any other supermarket", some customers may want the opportunity to shop for groceries, electronics, and wine. Here, what influences customers' choice of grocery store is the shortest route through one grocery store, one electronics store, and one wine shop rather than the shortest route to the grocery store alone. In this case, the RNN query needs to consider the influence of feature types besides grocery store. As the above example shows, there is a need to also consider the influence of other feature types in addition to that of the given query point.

After formalizing the MTRNN problem, we propose an on-line algorithm consisting of three major steps, preparation, filtering, and refinement. The preparation step finds feature routes for the filtering step by applying a greedy algorithm that uses R-tree indexes from all feature types during searching. The filtering step eliminates R-tree nodes that cannot contain an MTRNN by utilizing feature routes and then retrieves all remaining points that are potential MTRNNs to form a candidate MTRNN point set. We describe two pruning techniques, closed region pruning and open region pruning, to eliminate all R-tree nodes and points that cannot possibly be MTRNN points. The refinement step removes all the false hit points by three refinement approaches among which the final approach is to search the MTNN of each candidate point. Our experiments on both synthetic and real data sets demonstrate that typical MTRNN queries can be answered by our algorithms within reasonable time.

1.1 Overview

Chapter 2 addresses the MTNN search problem. We first discuss the motivation of the work and formalize the MTNN problem. Then we present an R-tree based page level pruning technique, called page level upper bound (PLUB) pruning, to prune the irrelevant R-tree nodes. The remaining points are further filtered to find the optimal solution by using the point level algorithm. Next, we compare the difference of our method with the RLORD algorithm, using a specific example and cost model. We then discuss the experiment results, showing the strength and weakness of our MTNN algorithm.

Chapter 3 extends the MTNN search problem to spatio-temporal road networks. We first formalize the BESTMTNN problem. Next we identify the special properties related to a BESTMTNN query and describe a special case of time-aggregated encoded path view which we call the Time-Aggregated Multi-Type Graph (TAMTG); we then present our partial route growth approach designed to accommodate the BESTMTNN query as well as a TAMTG-based label-correcting algorithm that finds the optimal solution for the BESTMTNN problem. Finally, experimental setup and experimental results are presented to show the computational performance of the BESTMTNN query algorithm on spatio-temporal road networks.

In Chapter 4, we introduce the new notion of the MTRNN query. We first formalize the MTRNN problem and present a brute force algorithm as a baseline algorithm. Following the definition of the MTRNN problem, we propose two filtering methods, closed region pruning and open region pruning, to prune the search space and three refinement approaches to remove the false hit points. We prove that our pruning and refinement approaches never introduce any false hit and false miss. Next, we formally analyze the complexity of the algorithms by presenting an analytical cost model. The experiment results show that the filtering and refinement based algorithms are several magnitudes faster than brute-force alternatives and give query results in reasonable time.

Chapter 2

Multi-Type Nearest Neighbor Search : Concepts and Algorithms

Given a query point and a collection of spatial features, a multi-type nearest neighbor(MTNN) query finds the shortest tour for the query point in a way such that only one instance of each feature is visited during the tour. For example, a tourist may be interested in finding the shortest tour which starts at a hotel and passes through a post office, a gas station, and a grocery store. The MTNN query problem is different from the traditional nearest neighbor query problem in that there are many objects for each feature type and the shortest tour should pass through only one object from each feature type. In this chapter, we propose an R-tree based solution exploiting a page level upper bound for efficient computation in clustered data sets and finding optimal query result. We compare our method with another recently proposed method, RLORD, which was developed to solve the optimal sequenced route(OSR) query [54]. In our view, OSR represents a spatially constrained version of MTNN. Experimental results are provided to show the strength of our proposed algorithm and design decisions related to performance tuning.

2.1 Introduction

Widespread use of spatial search engines such as Google Maps and MapQuest is leading to an increasing interest in developing intelligent spatial query techniques.

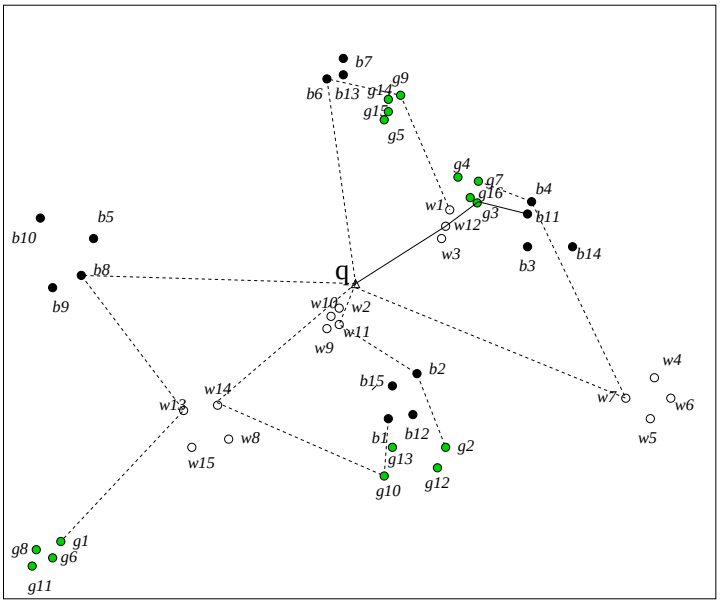


Figure 2.1: Multi-type nearest neighbor illustration

For example, a traveler may be interested in finding the shortest tour which starts at a hotel and passes through a post office, a gas station, and a grocery store. Therefore, it is critical to design an intelligent map query technique to efficiently find such a shortest tour. In this chapter, we formalize the above intelligent map query problem as a multi-type nearest neighbor (MTNN) query problem. Specifically, given a query point and a collection of spatial features, a MTNN query finds the shortest tour for the query point such that only one instance of each feature type is visited during the tour.

In the real world, many spatial data sets include a collection of instances of spatial features (e.g. post office, grocery store, and hotel). Figure illustrates an MTNN query. In the figure, points with different colors represent different spatial feature types. Given the query point **q** and a collection of spatial events represented by black(b) points, white(w) points and green/gray(g) points, an MTNN query is to find the shortest tour that starts at point **q** and passes through only one instance of each spatial event in the collection as the shortest route shown in the Figure 2.1. In this figure, the solid line string route (q, w_{12}, g_3, b_{11}) is a shortest path. All other dashed line strings represent alternative routes from q through one point from each feature type.

The nearest neighbor (NN) query problem [12,49,10,45,5,23,61] has been studied

extensively in the field of computer science. A traditional NN query can be stated as follows: given a point set $P = \{p_1, p_2, ...p_n\}$ and a query point q in a vector space, the NN query finds a point p_k such that the distance from q to $p_k \in P$ is minimized among the distances from q to $p_i \in P$. Many application domains are related to the NN query. For example, in Geographic Information Systems (GIS), "find the nearest gas station from my location" is a typical query that uses a NN query technique. In addition, NN queries are used for some data analysis techniques such as clustering.

Recently, many other NN query problems have attracted great research interests. All nearest neighbor (ANN) query [11, 13, 24, 25, 75] searches a nearest neighbor in a dataset A for every point in a dataset B. K-closest pair query [15, 14, 24, 25] discovers K-closest pairs within which a different point comes from a different dataset. Reverse nearest neighbor (RNN) query [31, 32, 58, 59, 17] finds a set of data that is the NN of a given query point. Group nearest neighbor (GNN) query [44] retrieves a nearest neighbor for a given set of query points. All of these problems focus on one or two data types and try to find relationships among data points within one or two object types. However, for many application domains, it is the relationship among more than two types of objects that's important.

The MTNN problem can have many variations if spatial and/or time constraints are imposed on it. For instance, we may constrain the range of selected object set PO within a given circle or rectangle, and the path can be from a query point q to all points in PO and return to q. If we know the visit order for part or all of the different feature types, it is a (partially) fixed order MTNN problem. Time constraint can also be part of the problem. For example, the post office might be open from 9:00am to 5:00pm so a visit has to be made during this period. However, our focus is on the generalized MTNN problem.

In this chapter, we study a generalized MTNN query problem and provide an optimal solution to the problem. Based on an R-tree index, we design an algorithm which exploits a page-level upper bound(PLUB) for efficient pruning at the R-tree node level. We originally formalized the MTNN query problem and presented algorithms for both optimal results and sub-optimal results in a technical report [39]. These algorithms are based on a page-level pruning strategy. In contrast, algorithms proposed for the OSR problem [54] apply instance-level pruning techniques for reducing the computation cost. In fact, the R-tree page-level pruning method can serve as a nice complimentary technique to the instance-level pruning method, since R-tree

page-level pruning technique makes better use of the R-tree index for reducing I/O cost. After discussion of our PLUB pruning strategy, we will give a detailed comparison of our method and the RLORD method, one of the solutions proposed by [54] for the OSR problem, introduced in [54]. Finally we give experiment results for both our method and the RLORD algorithm on clustered data sets.

Related Work. Previous work on NN can be classified in two groups. One consists of the main memory algorithms that are mainly proposed in computational geometry. The other is the category of secondary memory algorithms using R-tree index.

The simplest brute force algorithm can find a NN in $O(n)$ time. In the early period the main memory algorithms focused on developing efficient algorithms for datasets with specific distributions. Cleary analyzed algorithms on a uniformly distributed dataset that partition the space into a regular grid in [12]. Bentley *et al.* used k-d tree to get an $O(n)$ space and $O(log(n))$ time query result [20]. Another partition based approach [47] used the well-known Voronoi graph. It first precomputed the Voronoi graph for the given dataset. For a given query point q, it just needed to use a fast point location algorithm to determine the cell that contained the query point q.

The first R-tree based algorithm [49] for the NN query problem was a branch-and-bound algorithm in that it searches the R-tree using a depth first strategy and prunes the search space with the NN found so far. It basically uses two metrics, the MINDIST and MINMAXDIST, to prune the impossible R-tree node in the search as soon as possible. MINDIST is the distance from query point q to an object O and MINMAXDIST is the minimum of the maximum possible distances from p to a face of the Minimum Bounding Box(MBR) containing the object O.

The R-tree search begins at a root node downward to the leaf node. When necessary, the search will be upward. In a downward search, all MBRs with a MINDIST greater than the MINMAXDIST of another MBR will be discarded. In an upward search, an object with a distance to query point q greater than the MINMAXDIST of query point q to a MBR will be discarded and the MBR with a MINDIST greater than the distance from query point q to an object is also discarded.

Hjalason *et al.* employed a priority queue to implement a best first search strategy in [25]. This algorithm is optimal in the sense that it visits only the nodes along the path from the root to the leaf node that contains the NN.

Our proposed algorithm needs to find the MTNN from the remaining subsets

each of which contains at least one object of different types after reaching the leaf node. This is similar to the traveling salesman problem (TSP) [48], which tries to find the shortest path from a given dataset such that every data object is visited exactly one time. If the object number in feature types is limited to one, the MTNN query problem becomes a TSP problem. TSP is a NP-complete problem and the best known algorithms to find an optimal solution are exponential.

In parallel with our work, Sharifzadeh *et al.* [54] recently proposed an Optimal Sequenced Route (OSR) query problem and provided three optimal solutions: Dijkstra-based, LORD and R-LORD. Essentially, the OSR problem is a special case of the MTNN problem investigated in this chapter. Indeed, the OSR problem can be thought of as imposing a spatial constraint on the MTNN problem. Specifically, the visiting order of feature types is fixed for the OSR problem.

Another recently published work [36] proposed a number of fast approximate algorithms to give sub-optimal solutions in metric space for Trip Planning Queries(TPQ); this is the same type of query we call a MTNN query in the chapter.

Outline. The remainder of this chapter is organized as follows. Section 2.2 formalizes the MTNN problem. Section 2.3 presents an R-tree based optimal solution for the MTNN problem. Section 2.4 compares the difference of our method with the RLORD algorithm, using a specific example. The experimental setup and experiment results are provided in Section 2.5. Finally, in Section 2.6, we conclude our discussion and suggest further work.

2.2 Problem Formulation

In this section, we introduce some basic concepts, describe some symbols used in the rest of the chapter and give a formal problem statement for the MTNN query problem.

Let $< P_1, P_2, ..., P_k >$ be an ordered point sequence and $P_1, P_2, ..., P_k$ be from k different (feature) types of data sets. $R(q, P_1, P_2, ..., P_k)$ is a route from q though points $P_1, P_2, ...,$ and P_k and $d(R(q, P_1, P_2, ..., P_k))$ represent the distance of route $R(q, P_1, P_2, ..., P_k)$. Similarly, with R_i representing the tree node of feature type i we define a page-level upper bound(PLUB) as $d(R(q, R_1, R_2, ..., R_k))$, the longest distance of route $R(q, R_1, R_2, ..., R_k)$.

Multi-Type Nearest Neighbor (MTNN) is defined to be the ordered point sequence

$< P'_1, P'_2, ..., P'_k >$ such that $d(R(q, P'_1, P'_2, ..., P'_k))$ is minimum among all possible routes. Thus, $d(R(q, P'_1, P'_2, ..., P'_k))$ is the MTNN distance. An MTNN query is a query finding MTNNs in given spatial data-sets.

The following descriptions characterize a formal definition for the MTNN query problem.

Problem: The Multi-type Nearest Neighbor (MTNN) Query
Given:

- A query point, distance metric, k feature types of spatial objects and R-tree for each data set

Find:

- Multi-type Nearest Neighbor (MTNN)

Objective:

- Minimize the length of route from a query point covering an instance of each feature

Constraints:

- Correctness: The tour should be the shortest path for the query point and the given collection of spatial query feature types.
- Completeness: Only the shortest path is returned as the query result.

2.3 R-Tree Based Page-Level Pruning Algorithm

In spatial databases, R trees and theirs variants are widely used for indexing spatial data. In this chapter, we propose an R-tree based algorithm for the MTNN query problem. Specifically, we design an R-tree based page-level pruning method to filter out large numbers of spatial objects. This method gives an optimal solution and has exponential time complexity with respect to the number of feature types. The algorithm works well when the number of feature types is small (< 8).

We have many feature types in an MTNN problem. In order to find the optimal solution, we have to search a space consisting of all permutations of all feature type objects. For every permutation, we do the same search steps and get a route with a

shortest distance. Thus for total N permutations, we get N routes. Finally we find the solution to the MTNN problem by taking the route with the shortest distance from these N routes. For the sake of convenience, our discussions are based on a search space consisting of one permutation of all feature type objects in the following.

For one permutation of feature types $t_1, t_2, \ldots, t_k$, we need to find the optimal route from the query point through one point in every type in the order of $t_1, t_2, \ldots, t_k$. In the R-tree based algorithm we use a branch and bound strategy to prune and search the space. The algorithm can be divided into three parts. The first part finds an upper bound for the R-tree search. The second part prunes the search space based on R-tree using the current upper bound. The output of this part is candidate sequences consisting of leaf nodes, each of which is from one of the R trees. The third part finds the current MTNN shortest distance from the current candidate sequence. Figure 2.2 illustrates these three parts. We will discuss them in detail in the rest of this section.

Algorithm MTNN(R-trees, q)
Input : K types of spatial objects and R-tree, Distance metrics,
the query point q
Output : MTNN and the shortest path
1. **step 1: First Upper Bound Search** Find the first upper bound of
2. MTNN shortest distance by using a fast greedy algorithm and set
3. current upper bound to be this first upper bound
4. **step 2: R-Tree Search** Prune search space to find subsets of objects
5. that may contain MTNN and get a candidate sequence
6. **step 3: Subset Search** Calculate current MTNN shortest distance in
7. current candidate sequence
8. **if** current calculated MTNN shortest distance shorter than
9. current upper bound
10. **then** set current upper bound to be current calculated MTNN
11. shortest distance
12. **if** Some search space is not examined
13. **then** Go to step 2
14. **else** Report current upper bound as the final MTNN shortest
15. distance

Figure 2.2: R-tree based MTNN algorithm

2.3.1 First Upper Bound Search

The first step of the MTNN algorithm is to find the first upper bound for pruning the search space. This upper bound will determine the pruning efficiency for the R-tree search. The general requirements for the first upper bound search strategy are time efficiency and upper bound accuracy. Trade-offs will be made when designing an MTNN algorithm. In most cases, we prefer an algorithm with high time efficiency and normal upper bound accuracy. In this chapter, we use a simple greedy algorithm as follows.

Randomly generate one permutation of feature types, for example, generate permutation $R = (r_1, r_2, \ldots, r_k)$. Search the NN r_{1,i_1} of query point q in feature type r_1 by using a basic R-tree based NN search method. Then search the NN r_{2,i_2} of r_{1,i_1} in feature type r_2. Repeat this procedure until all types of features are visited. Finally, we get a path from query point q going through an exact single point in each feature type. Calculate the distance of this path and use it as the first upper bound in the MTNN search. We call this distance the greedy distance r_g.

2.3.2 R-Tree Search

In spatial databases, the task of an R-tree search is to prune the search space using a branch and bound approach on the R-tree index. We call the pruning method used in this part R-tree page-level pruning. For permutation $R = \{r_1, r_2, \ldots, r_k\}$ we first use a general NN search strategy to determine in the R-tree of type r_1 the possible leaf node rectangle set S_1 such that (d, R_{s1}) $(R_{s1} \in S_1)$ is less than the upper bound distance. Next the rectangle set S_1 is used to determine the possible leaf node rectangle set S_2 in the R-tree of type r_2 such that the distance $d(q, R_{s1}, R_{s2})$ $(R_{s1} \in S_1,\ R_{s2} \in S_2)$ is less than the upper bound distance. This procedure continues until all R-trees are visited. Finally, we get a list of candidate leaf node sequences among which each leaf node contains one type of feature objects. When searching R trees we choose to use a Depth First Search(DFS) strategy since DFS generates a route distance faster and we may use the new generated route distance as an upper bound if it is shorter than the current upper bound and thus prune R-tree nodes more efficiently.

2.3.3 Subset Search

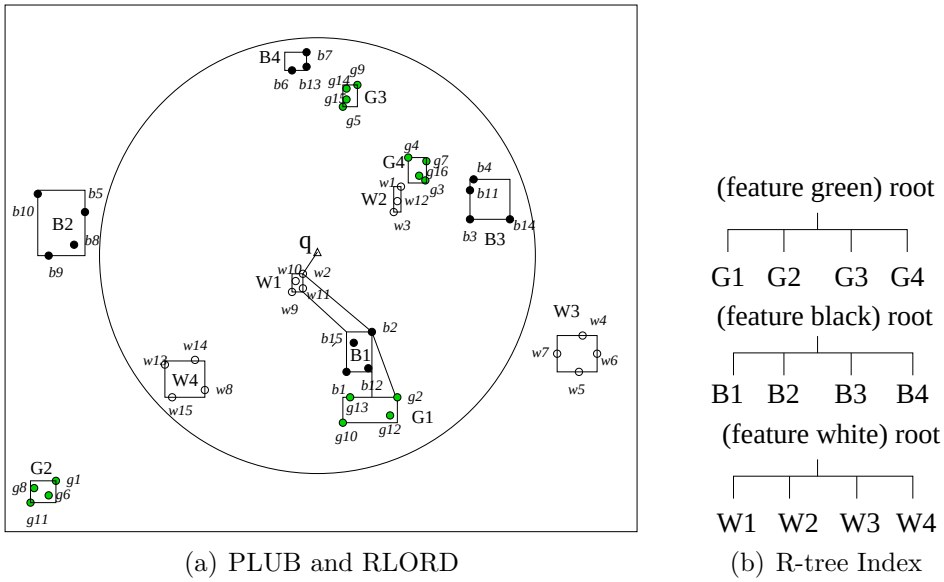
In a subset search, we are given subsets of all different types of objects for all permutations of different feature types. For a specific permutation, all these points in

subsets form a multi-level bipartite graph. The legal route consists of points each of which is from a different level of the graph. Many search algorithms such as BFS, DFS, $Dijkstra$, A^*, IDA^*, SMA^* etc can be updated and used to find the optimal route. We call the methods used in this part point pruning. In [39], a simple brute force algorithm and a dynamic programming method were given. In this chapter, we use the RLORD algorithm [53] as another search method in our subset search.

2.4 Comparison of PLUB and RLORD

2.4.1 Comparison by Example

Here, we illustrate our proposed PLUB-based MTNN algorithm and compare it to R-LORD by using an extended example from [54]. Basically, a MTNN problem reduces to an OSR problem for a fixed permutation of feature types. The following discussions are based on a fixed permutation.



(a) PLUB and RLORD (b) R-tree Index

Figure 2.3: A running example for PLUB and RLORD

In the example of Figure 2.3 (a) we assume the permutation is (w, b, g) and the distance metric is the Euclidian distance. The order of the R-tree is 4. There are three different feature types represented by black(b), white(w) and green/gray(g) points. In Figure 2.3 (a), $R(q, w_2, b_2, g_2)$ is the greedy route and the radius of the search circle is $d(R(q, w_2, b_2, g_2))$. $\mathbf{q}$ is the query point represented as $\triangle$ and the rectangles

represent the leaf nodes of the R-tree indices for different feature types. Figure 2.3 (b) gives the R-tree structure for feature types green, black and white.

The first step in PLUB is the same as in R-LORD: look for the first upper bound distance. The algorithm first finds NN w_2 of q in all objects of feature type w. Then b_2 of feature type b is found as the NN of w_2. Next, g_2 of feature type g is found as the NN of b_2. Finally we get greedy route R_g (q, w_2, b_2, g_2) with greedy distance $D_g = d(R(q, w_2, b_2, g_2)) = 3.37$ as the current upper bound D_u.

In the R-tree search, leaf node W_1 is inside the upper bound circle, so the partial route is expanded to be (q, W_1). Next, the R-tree of feature type b is searched, and leaf nodes B_1, B_3, B_4 are added to the current partial route (q, W_1) because the PLUB of partial routes (q, W_1, B_1), (q, W_1, B_3) and (q, W_1, B_4) is less than the current upper bound. Then we search the R-tree of feature type g and find that the PLUB of only one route (q, W_1, B_1, G_1) is less than the current upper bound. Thus in the subset search step, we only need to look for the shortest route from query point q through points inside leaf nodes W_1, B_1 and G_1. Table 2.1 gives the detailed calculation results.

			Upper Bound	Eliminated
W_1	B_1	G_1	2.04	N
W_1	B_1	G_3	6.2	Y
W_1	B_1	G_4	4.27	Y
W_1	B_3	G_1	7.53	Y
W_1	B_3	G_3	6.54	Y
W_1	B_3	G_4	4.29	Y
W_1	B_4	G_1	4.02	Y
W_2	B_1		3.7	Y
W_2	B_3	G_4	3.43	Y
W_2	B_4		5.17	Y
W_4	B_1		4.08	Y
W_4	B_3		7.94	Y
W_4	B_4		7.56	Y

Table 2.1: Calculation Results of PLUB Leaf Node Sequences

When searching candidate MTNNs in route $R(q, W_1, B_1, G_1))$, the first iteration does 4 point-to-point$(P - P)$ calculations and finds partial routes $R(q, g_2)$, $R(q, g_{10})$, $R(q, g_{12})$ and $R(q, g_{13})$. Similarly, iteration 2 gives partial routes $R(q, b_{12}, g_{13})$, $R(q, b_1, g_{13})$, $R(q, b_2, g_2)$ and $R(q, b_{15}, g_{13})$ with 20 $(P - P)$ calculations. Finally we get

$R(q, w_{10}, b_{15}, g_{13})$, $R(q, w_9, b_{15}, g_{13})$, $R(q, w_2, b_2, g_2)$, and $R(q, w_{11}, b_1, g_{13})$ with 20 P-P calculations. After this step, the current MTNN is $R(q, w_{11}, b_1, g_{13})$ with distance 3.16. This procedure takes 44 total P-P calculations.

In R-LORD, initially the partial route set is $S = \{(g_2), (g_3), (g_4), (g_5), (g_7), (g_9), (g_{10}), (g_{12}), (g_{13}), (g_{14}), (g_{15}), (g_{16})\}$. In the first iteration, every black point x inside T_c (range query Q1) and MBR(Q2) (range query Q2) is checked for every green/gray point in S. If $D(p, x) + D(x, P_1) + L(PSR) \leq T_c$, then point x is added to the head of the partial route. When x is b_1, for example, we get partial route $(b_1, g_{10}), (b_1, g_{13})$. By using property 2, only partial routes with shortest length will be kept. So, (b_1, g_{13}) is put into a new partial route set. At the end of iteration 1, we have partial route set $\{(b_1, g_{13}), (b_2, g_2), (b_3, g_3), (b_4, g_3), (b_6, g_{14}), (b_7, g_{14}), (b_{11}, g_3), (b_{12}, g_{13}), (b_{13}, g_{14}), (b_{14}, g_3), (b_{15}, g_{13})\}$. By using property 2, we dramatically reduce the size of the partial route set. However, property 2 can only be used in iteration 1. Following a similar procedure, each of subsequent $(m - 2)$ iterations will check every point of the feature type inside T_v (range query Q1) and MBR(Q2) (range query Q2) for every partial route in the current partial route set S. Finally we get route set $\{(w_1, b_{11}, g_3), (w_2, b_2, g_2), (w_3, b_{11}, g_3), (w_8, b_1, g_{13}), (w_9, b_{15}, g_{13}), (w_{10}, b_{15}, g_3), (w_{11}, b_1, g_{13}), (w_{12}, b_{11}, g_3), (w_{13}, b_1, g_{13}), (w_{14}, b_1, g_{13}), (w_{15}, b_1, g_{13})\}$ and $R(q, w_{11}, b_1, g_{13})$ is shortest among all routes. This procedure takes a total of 298 P-P calculations.

In summary, PLUB needs 17 rectangle-to-rectangle distance calculations and 44 $P - P$ distance calculations in this example. RLORD takes 298 $P - P$ calculations. Apparently PLUB requires less computation.

As seen in the above illustration, the PLUB method uses a page-based pruning approach, while R-LORD uses a point-based search method. If the number of points inside the query range in R-LORD becomes big, the size of the partial route set will increase significantly. For every partial route inside a current partial route set, every point of the following feature type inside the current query range in R-LORD needs to be checked, which takes a lot of time.

2.4.2 Comparison by Cost Models

In this section, we provide algebraic cost models for PLUB and RLORD. The MTNN query is a CPU intensive task, and the CPU cost is at least as important as the I/O cost for data sets with medium and high numbers of feature types of spatial data.

We will explore the I/O cost model and give a whole cost analysis for PLUB and RLORD.

As we discussed in section 3, the costs for the proposed PLUB include (1) the search of the R-tree leaf nodes inside the current search range, (2) page-level leaf node candidate sequence pruning and (3) a point-level candidate MTNN search. Therefore, the cost model also has three components: (1) cost of the page-level R-tree traversal, (2) cost of the page-level leaf node candidate sequence search, and (3) cost of the point-level candidate MTNN search. We also identify the cost components of RLORD as (1) cost of the page-level R-tree traversal and (2) cost of the point-level candidate MTNN search. In PLUB, the page-level leaf node candidate sequence search will possibly prune many more candidate sequences so the cost of the point-level candidate MTNN search will possibly be much smaller than that in RLORD. In the following, we will discuss the cost models for PLUB and RLORD respectively.

Cost Model of PLUB

Let C_{R-T} be the cost of the R-tree traversal to find all R-tree leaf nodes intersected by the circle with radius of the current upper bound, centered at the query point. In addition, let C_{LF} be the page-level leaf node search cost for the R-tree candidate leaf node sequences and C_{PN} be the point-level search cost for candidate MTNNs in candidate leaf node sequences. Thus the total cost of PLUB is expressed as $C_{R-T} + C_{LF} + C_{PN}$.

PLUB first traverses all the R-trees to look for leaf node rectangles that intersect with the current search range. Let C_{PR} be the cost of the point-to-rectangle distance calculations and $N_{t,i}$ be the number of all the tree nodes visited in the feature type i tree traversal. Thus $C_{R-T} = C_{PR} \times \Sigma N_{t,i} (i = 1, ..., k)$ (k is the number of feature types). It is worth noting that C_{R-T} is the same for PLUB and RLORD.

Next PLUB does a page-level search for leaf node candidate sequences. Let N_{R-R} be the number of leaf nodes visited in candidate leaf node sequences, and C_{R-R} be the cost of rectangle-to-rectangle distance calculation. Then we can get the cost of leaf node candidate sequence pruning as $C_{LF} = N_{R-R} \times C_{R-R}$.

Finally we search for candidate MTNNs in the remaining leaf node candidate sequences. Let F_{LS} be the leaf node candidate sequence filtering ability ratio, n_l be the average point number in leaf node for all feature types and p_i be the page number of feature type i. We use C_{ls} to denote the cost of the MTNN search in single leaf

node sequence to arrive at the following cost:

$$C_{ls} = n_l + (n_l \times n_l) + n_l + (n_l \times n_l) + ... + n_l + (n_l \times n_l) \text{ } (k-1 \text{ items})$$

The condensed form is:

$$(k-1)(n_l \times (n_l + 1))$$

Thus the total point-level candidate MTNN search cost is $C_{PN} = C_{ls} \times \Pi p_i \times (1 - F_{LS}), (i = 1, ..., k)$

Cost Model of RLORD

Let C_{R-T} be the cost of R-tree based coarse pruning, i.e., finding all data points inside the initial upper bound, and let C_{PS} be the cost of the candidate MTNN search in the remaining subsets. The cost C_{R-T} is the same as for PLUB. The cost C_{PS} is:

$$n_l \times (p_1 + n_l \times p_1 \times p_2 + (p_2 + n_l \times p_2 \times p_3) + ... + (p_{k-1} + n_l \times p_{k-1} \times p_k)$$

The total cost of RLORD is $C_{R-T} + C_{PS}$.

The Cost Model Comparison of PLUB and RLORD

There are many factors to consider when comparing the cost models of PLUB and RLORD. We may consider simplifying these models by focusing on the dominant factors and therefore removing some terms. In PLUB we may assume that $C_{R-T} + C_{LF} << C_{PL}$ and get the approximate cost model as:

$$(k-1)n_l \times (n_l + 1) \times \Pi p_i \times (1 - F_{LS}).$$

Similarly, if we assume $C_{R-T} << C_{PS}$, R-LORD's cost model becomes:

$$n_l \times (p_1 + n_l \times p_1 \times p_2 + (p_2 + n_l \times p_2 \times p_3) + ... + (p_{k-1} + n_l \times p_{k-1} \times p_k)$$

In random or approximate random datasets, F_{LS} is small, and PLUB takes more time. The opposite is true in clustered datasets, where F_{LS} tends to be bigger. That is when

$$1 - F_{LS} < n_l \times (p_1 + n_l \times p_1 \times p_2 + (p_2 + n_l \times p_2 \times p_3) + ... + (p_{k-1} + n_l \times p_{k-1} \times p_k))/((k-1)n_l \times (n_l + 1) \times \pi p_i)$$

PLUB runs faster than RLORD. Later in the discussion of the experimental results, we'll refer to this formula 1, and refer to the left side as the remaining ratio ($r-ratio$), and the right side as the comparison ratio ($c - ratio$).

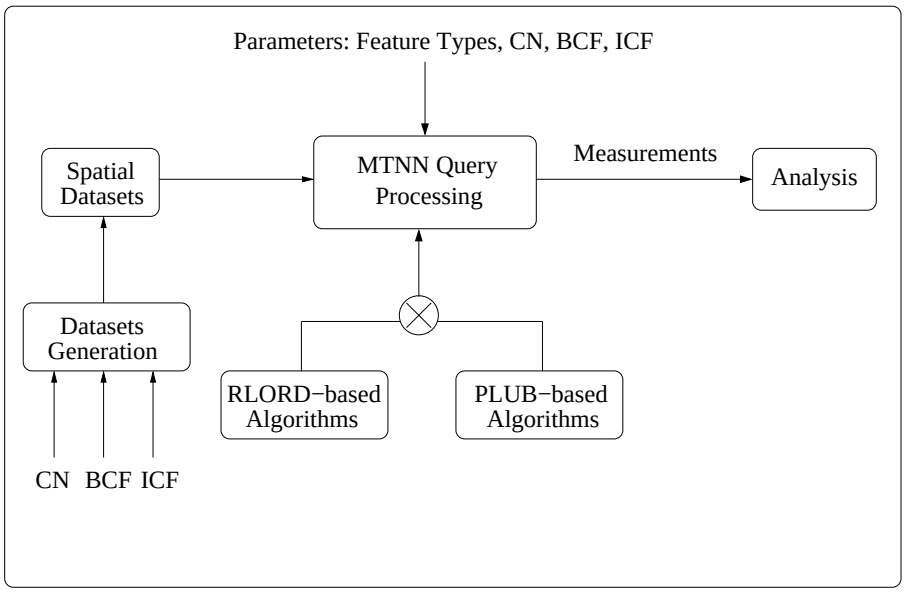


Figure 2.4: Experiment setup and design

2.5 Experimental Results

In this section, we present the results of various experiments to evaluate our PLUB based algorithm and RLORD based algorithm, both of which give optimal solutions, for the MTNN query in different clustered data sets. Specifically, we demonstrate comparisons of the PLUB and RLORD based algorithms with respect to execution time under different data sets with different properties such as feature type number, data set density and compactness of clusters.

2.5.1 The Experimental Setup

Experiment Platform Our experiments were performed on a PC with a 3.20GHz CPU and 1 GByte memory running the GNU/Linux Ubuntu 1.0 operating system. All algorithms were implemented in the C programming language.

Experimental Data Sets We evaluated the performance of both the PLUB and RLORD based algorithms for the MTNN query with synthetic data sets, which allow better control towards studying the effects of interesting parameters. All data points in the synthetic data sets were distributed over a 10000X10000 plane and formed clustered data sets. In order to reduce the effect of query point positions, we took 25 query points on a sample dataset space, each of whose x and y axis values were from 3000.00 to 7000.00 respectively and with each point placed 1000.00 away

from its neighbor in the x and y axis directions, and calculated the average running time, $c - ratio$ and $r - ratio$ as the final reported values. There were four different parameters in our experimental setup.

- Feature Type(FT): Feature type numbers from 2 to 7 to show the scalability of both algorithms.
- Between-cluster Compactness Factor(BCF): control the minimum distance of cluster centers, i.e. the compactness between clusters.
- In-cluster Compactness Factor(ICF): control the compactness within a cluster.
- Cluster Number(CN): control the of density of data sets.

For a given cluster number ClusterNumber, we generated a data set as follows. First a simplified estimated maximum number of cluster center distance was determined by formula $maxCCDist = 10000.0/(int)(\sqrt{ClusterNumber} + 1)$. Next the minimum cluster center distance was calculated as follows $minCCDist = BCF \times maxCCDist$. Finally, we decided the cluster size by $ClusterSize = ICF \times minCCDist$ The number of objects inside each cluster is within $p/2$ and p, 84 in our experiment setting, that is the order of R-tree leaf node. Thus the expected number of objects inside a single cluster is about 61. For a dataset of 20 clusters, the total object number is therefore about 1220.

Experiment Design Figure 2.4 describes the experimental setup to evaluate the impact of design decisions on the relative performance of both the PLUB and RLORD based algorithms for the MTNN query. We evaluated the performance of the algorithms with synthetic data sets generated according to the rules discussed above. We observed the performance of both PLUB and RLORD based algorithms under different data set settings in term of execution time. Our goal was to answer the following questions: (1) How do changes in feature type affect scalability in PLUB and RLORD? (2) How do differences in data density affect the performance of PLUB and RLORD? (3) How does compactness between clusters affect the performance of PLUB and RLORD? (4) How does compactness within clusters affect the performance of PLUB and RLORD?

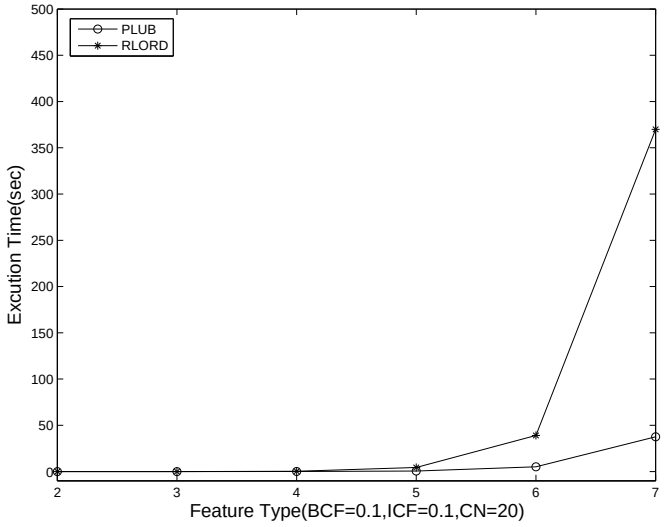


Figure 2.5: Scalability of PLUB and RLORD in terms of feature types

2.5.2 A Performance Comparison of PLUB and RLORD With Different Feature Types

This section describes the scalability improvement of PLUB in terms of feature types in clustered data sets, compared to RLORD. We set the fixed cluster number at 20, the BCF at 0.1, which means the minimum cluster center distance was 10% of maxCCDist and the ICF at 0.1, which means the size of a cluster was 10% of the minimum distance between two clusters. This is a highly clustered dataset in that the size of clusters is 1% of maxCCDist. We change the number of feature types from 2 to 7 and don't show the results with feature type number 1 because that case would reduce the MTNN query problem to the classic NN problem, making PLUB and RLORD no more than classic NN algorithms.

Figure 2.5 compares the scalability of PLUB and RLORD in terms of numbers of feature types. More specifically, this figure illustrates that the execution time change with the increase of data types from 2 to 7 when the minimum distance between clusters is small (BCF=0.1) and cluster size is small (ICF=0.1) When the data type number is 2,3,4 and 5, there is no big difference of performance between PLUB and RLORD. When the data type number is 6 and 7, PLUB runs less time than RLORD. This experiment shows that PLUB is more scalable than RLORD in highly clustered data sets.

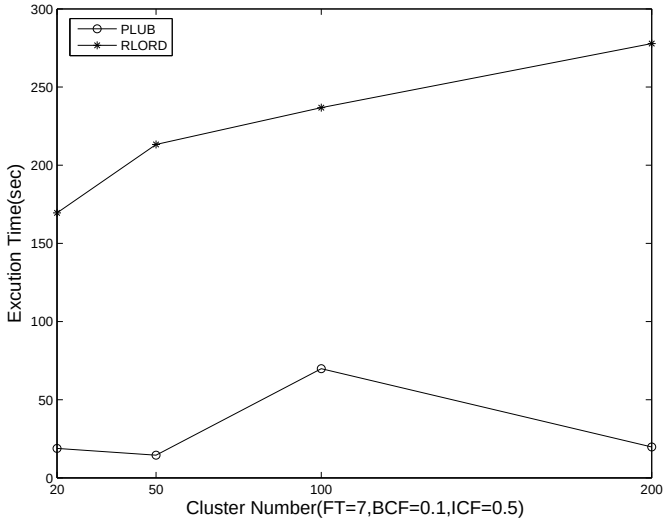


Figure 2.6: Performance of PLUB and RLORD on different densities of data sets

2.5.3 The Effect of Data Set Density On Performance of PLUB and RLORD

In this section, we show how the density of data sets affects the performance of PLUB and RLORD. We tested PLUB and RLORD with feature type number 7, BCF 0.1, which is the same BCF used in the scalability test, and ICF 0.5, which means the size of a cluster was 50% of the minimum distance between two clusters. The changing variable is cluster number, with assigned values of 20, 50, 100 and 200. Because there are almost the same average number of data points inside clusters for data sets of different cluster numbers, these data sets on the same space represent data sets with different densities.

Figure 2.6 illustrates the performance of PLUB and RLORD on different densities of data sets. As can be seen, under all dataset densities with cluster numbers 20, 50, 100 and 200, the execution time of PLUB is always less than RLORD. In this figure we cannot see significant change in execution time, or any apparent trend for either PLUB and RLORD, which means the data set density appears to have almost no effect on execution time of PLUB and RLORD in clustered data sets with current settings.

2.5.4 Effect of Between-Cluster Compactness Factor on Performance of PLUB and RLORD

In this section, we show the effect of the between-cluster compactness factor (BCF) on the performance of PLUB and RLORD. We set the feature type number at 7, ICF

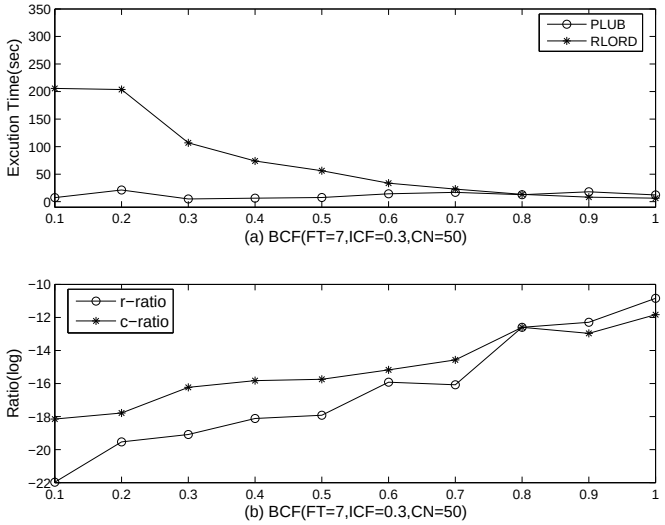


Figure 2.7: Effect of between-clusters compactness factor

at 0.3 and cluster number at 50, which is medium density in our experiments. We raised parameter BCF from 0.1 to its highest value 1.0.

Figure 2.7(a) illustrates the performance of PLUB and RLORD on data sets with different BCF. We can see that both the execution times and the trends of PLUB and RLORD are very different. The execution time of RLORD has an apparent down trend with the increase of BCF from 0.1 to 1.0. However, the execution time of PLUB doesn't change too much. With BCF values smaller than some value, about 0.8 in this specific experimental settings, PLPUB runs faster. When BCF increases beyond this value, RLORD is faster.

Figure 2.7(b) gives the results of formula 1. The curve $r - ratio$ shows the ratio of the left side of formula 1 and the curve $c - ratio$ presents the ratio of the right side of formula 1. Both ratio values are log values because they are tiny numbers, which means the pruning ability is very high. A seemingly contradictory result evident in this figure is that increases in the $r - ratio$, which means there is a decrease in the pruning ratio, does not lead to increases in execution time. The explanation is that when BCF increases, there are fewer leaf nodes that intersected with the current search bound. Thus the total number of possible candidate leaf node sequences decreases dramatically, thereby reducing the execution time. The key point to note here is that when the $r - ratio$ is smaller than the $c - ratio$, PLUB runs faster but when the remaining ratio is greater than the comparison ratio, PLUB takes more time than RLORD. In other words, the relative trends of $r - ratio$ and $c - ratio$ only

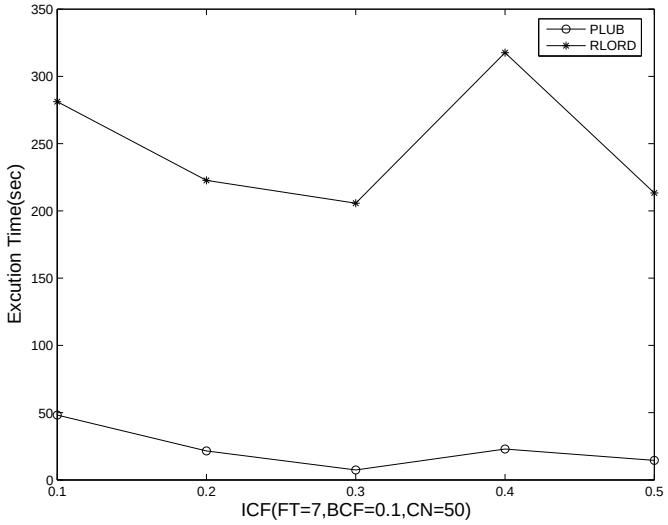


Figure 2.8: Effect of in-cluster compactness factor

determine the relative execution time of PLUB and RLORD.

2.5.5 Effect of In-Cluster Compactness Factor on Performance of PLUB and RLORD

In this section, we show the effect of the in-cluster compactness factor (ICF) on the performance of PLUB and RLORD. We set the feature type number at 7, BCF at 0.1 and cluster number at 50, or medium density. We changed parameter ICF from 0.1 to 0.5.

Figure 2.8 illustrates the performance of PLUB and RLORD on data sets with different ICF. We can see the execution times of PLUB and RLORD are very different. With $BCF = 0.1$, ICF has little influence on the execution time of either PLUB or RLORD, which means if the minimum allowed distance $minCCDist$ of clusters is very small, compared to the maximum allowed distance $maxCCDist$, in our experimental settings, the effect of BCF is dominant among all factors. From this figure the only apparent trend is that PLUB always runs much faster than RLORD under these experimental settings.

2.6 Summary

In this chapter, we investigated a multi-type nearest neighbor (MTNN) query problem, which can be related to many application domains, such as intelligent map

quest. We show that the MTNN problem is closely related to the TSP problem, but the computation complexity of the MTNN problem is much higher than that of the TSP problem in terms of feature type. We propose a R-tree based solution to MTNN query problem. In our algorithm, a page-level upper bound (PLUB) is exploited for efficient pruning at the R-tree node level. Finally, experimental results are provided to show the strength of the proposed algorithm and design decisions related to performance tuning. In our experiments, we compare the performances of PLUB and RLORD in terms of execution time. When data sets are compact, PLUB outperforms RLORD. When data sets go to random-distributed in space, RLORD runs faster than PLUB.

As for future work, we plan to investigate heuristic algorithms from different perspectives since MTNN query problem is very complex. For instance, one direction is to design heuristic algorithms using geometric properties of spatial data sets. Also, we believe that PLUB algorithm is very adequate to be extended to real road network due to PLUB's page-level pruning technique.

Chapter 3

Multi-Type Nearest Neighbor Query on Spatio-Temporal Road Networks

A multi-type nearest neighbor(MTNN) query finds the shortest tour for a given query point and different types of spatial features such that only one instance of each feature is visited during the tour. In a real life MTNN query a user normally needs an answer with specific start time and turn-by-turn route for specific period of time on road networks, which requires considerations of spatial and temporal features of the road network when designing algorithms. In this chapter, we propose a label correcting algorithm that is based on a time aggregated multi-type graph, a special case of a time aggregated encoded path view. This algorithm gives the best start time, a turn-by-turn route and shortest path in terms of least travel time for a given query. Experimental results are provided to show the strength of our proposed algorithm and design decisions related to performance tuning.

3.1 Introduction

Widespread use of spatial search engines such as Google Maps and MapQuest is leading to an increasing interest in developing intelligent spatial-temporal query techniques. For example, a traveler may be interested in finding a shortest route in terms

of least travel time with the best start time between 9:00 am and 11:00 am from his house through one grocery store (with a stay of 1 1/2 hours), one electronics store (1 hour stay) and one post office (arriving before 4:00 pm; 1/2 hour stay) and returning home before 8:00 pm. This query illustrates some important properties. First, the traveler is trying to find a route with instances from different feature types (grocery store, electronics store, etc.). This kind of query is called a multi-type nearest neighbor (MTNN) query in [40]. Second, the route to be found is a closed route from the query point back to the query point. Third, the traveler is interested in not only the route but also the best start time. Considering the variability of traffic patterns at different times on road networks, this best start time could differ for different time windows. Therefore, the query asks for answers containing not only spatial features like the route but also temporal features. Forth, the query itself contains spatial and temporal features. For example, the query point and different interested locations are spatial features. The best start time between 9:00 am and 11:00 am and length of stay at each location are temporal features.

In this chapter, we extend MTNN query in the temporal dimension and study the spatio-temporal MTNN query problem on spatial-temporal road networks. We formalize a common query in real life as a spatial-temporal MTNN query, called BEst Start Time Multi-Type Nearest Neighbor (BESTMTNN) query, with time window constraints and answer the query based on our extension of the encoded path view. This extension extends the encoded path view from spatial-only to spatio-temporal road networks and is called the Time Aggregated Multi-Type Graph (TAMTG), a special case of the time aggregated encoded path view (TAEPV) of road networks. By identifying the special properties of BESTMTNN query that lead to our spatia-temporal partial route growth approach we propose a label-correcting based algorithm to solve it. This algorithm prioritizes the spatial-temporal partial routes with current least travel time. It takes a user-specified query that involves spatio-temporal features such as query time window sizes for all features and planned stay time interval at a location and gives a turn-by-turn route and the best start time in terms of least travel time. Our experiments show our algorithm can answer normal user BESTMTNN queries in a reasonable time.

Related Work. Vehicle routing and scheduling problems have been extensively studied in Operational Research. According to a taxonomy given by Bodin *et al.* [7], vehicle routing involves the traversing of a sequence of points in order. Vehicle scheduling

involves traversing a sequence of points with an associated set of departure and arrival times. If a vehicle must traverse a sequence of points with time window and/or precedence relationships, the problem is a combined vehicle routing and scheduling problem. Numerous computational methods to solve such problems have been developed. Laporte *et al.* presented a summary of exact and approximate algorithms for vehicle routing and scheduling problems [33]. He also summarized classical and modern heuristics for solving such problems [34]. However, in all of these works, the sequence of points to be visited in a query was specified in advance. Thus no solution required that the point space be searched for points that would be visited in the query, a key difference between previous works and our MTNN query problem.

In order to quickly find answers to spatial queries, researchers normally model road networks as a graph. Huang *et al.* [26] precomputed all-pair shortest paths and stored them in a spatial database. This precomputed graph is called an Encoded Path View (EPV). Later, Huang *et al.* [27] extended EPV to large road networks and proposed a Hierarchical Encoded Path View (HEPV), which provides a great way to scale many algorithms to large road networks. Referencing a HEPV makes it possible to answer a nearest neighbor query on road network very efficiently.

Recently, George *et al.* [22] proposed a Time-Aggregated Graph (TAG) to model a spatial-temporal network. Based on this model, spatial queries that have been studied for decades are answered along both spatial and temporal dimensions. For example, George showed that the SP-TAG algorithm computes the shortest path for a given start time in a small time-dependent network. In a related study, Ding [19] proposed a time-dependent graph and studied how to find the best departure time in terms of least travel time from one place to another over a large road network.

Meanwhile, the queries related to multiple feature types attracted attentions from different database research groups. X. Ma *et al.* [40] formalized a MTNN query problem and proposed a Page Level Upper Bound (PLUB) based algorithm to find an optimal route for the MTNN query. Sharifzadeh *et al.* [54] recently proposed an Optimal Sequenced Route (OSR) query problem and provided three optimal solutions, Dijkstra-based, LORD and R-LORD, to solve the OSR query problem. Essentially, the OSR problem is a special case of the MTNN problem. Because it fixes the visiting order of feature types, it can be thought of as imposing a spatial constraint on the MTNN problem. Sharifzadeh *et al.* [55] extended the OSR work to road network by using Voronoi diagrams. Basically the algorithm of this extension precomputes

Voronoi diagrams for every possible partial route and finds the optimal sequenced route very efficiently. However, it does not consider the variation in traffic patterns that occurs at different times and thus ignores the temporal dimension of road networks. It also does not give a turn-by-turn route for the query on road networks. Another issue is that with Euclidean distance (i.e. L_2 norm) as metric the cell edges in Voronoi diagram may become hyperbolic curves, which makes the determination of whether or not a point is inside a cell very difficult.

Another recently published work [36] proposed a number of fast approximate algorithms to give sub-optimal solutions in metric space for Trip Planning Queries (TPQ). This work focused on efficient algorithms but could not guarantee finding the shortest path.

Outline. The remainder of this chapter is organized as follows. Section 3.2 formalizes the BESTMTNN problem. In section 3.3 we identify the special properties related to a BESTMTNN query and describes the Time-Aggregated Multi-Type Graph (TAMTG); we then present our partial route growth approach designed to accommodate the BEST-MTNN query as well as a TAMTG-based label-correcting algorithm that finds the optimal solution for the BESTMTNN problem. Section 3.4 gives the experimental setup and experimental results. Finally, in Section 3.5, we conclude our discussion and suggest further work.

3.2 Basic Concepts and Problem Formulation

In this section, we introduce some basic concepts, explain some symbols used in the remainder of the chapter and give a formal statement of the BEst Start Time Multi-Type Nearest Neighbor (BESTMTNN) query problem.

Let $< P_{1,1'}, P_{2,2'}, ..., P_{k,k'} >$ be an ordered point sequence and let $P_{1,1'}, P_{2,2'}, ..., P_{k,k'}$ be from k different (feature) types of data sets. P_{i,i',t_i} is a point P'_i of feature type i at time t_i. A spatial-temporal *Partial Route* $R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{l,l',t_l})$ is a route from the query point q at time t_0 through points from different feature types but not back to the original query point q. A complete closed route $R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, \ldots, P_{k,k',t_k}, q_{t_{k+1}})$ is a route that goes from the query point q at time t_0 through point $P_{1,1'}$, goes from point $P_{1,1'}$ at time t_1 through $P_{2,2'}$, ..., and returns to query point q at time t_{k+1} from $P_{k,k'}$ at time t_k. $t(R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t_{k+1}}))$ represents the travel time through the route $R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t_{k+1}})$.

A BESTMTNN is defined to be an ordered point sequence $< q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t'_0} >$ such that $t(R(q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t'_0}))$ is minimum among all possible routes for all possible start time points and all qualified time window. A BESTMTNN query is a query finding a BESTMTNN that includes a best start time, a turn-by-turn route, and a least travel time in given spatial and temporal data sets. Thus, $< q_{t_0}, P_{1,1',t_1}, P_{2,2',t_2}, ..., P_{k,k',t_k}, q_{t'_0} >$ is a BESTMTNN query result.

An Encoded Path View (EPV) stores all pairs of shortest distance paths in a spatial database. A Time Aggregate EPV (TAEPV) records such spatial information but extends the EPV to include temporal information. In other words, a TAEPV stores all pairs of shortest distance for all time points in terms of least travel time. With a TAEPV on road networks, the results of a BESTMTNN query consolidate the location information of interested points with different traffic patterns that vary by times. A Time Aggregate Multi-Type Graph (TAMTG) is a special case of a TAEPV. In a TAMTG, only least travel times among all points of interest (POI) are stored. In a BESTMTNN query, the POIs are the given k data sets from k different feature types.

The following is a formal definition of the BEst Start Time MTNN (BESTMTNN) query problem. In the BESTMTNN query problem, we use road distance since we are searching for BESTMTNN on spatial-temporal road networks. The spatial-temporal road networks are represented by a TAEPV and a TAMTG that store least travel times among points. These least travel times are calculated in advance because routing and scheduling are extremely time-consuming. Considering POIs could number from hundreds to thousands and even more, it is infeasible to do an ad-hoc routing and scheduling for every BESTMTNN query. In this problem, we also consider time window constraints as user-specified parameters indicating what time intervals qualify for this query and how long the user is going to stay at a location (planned stay period). Time window constraints are meaningful parameters in daily life. For example, a post office can be visited only certain hours in a day, and a movie-goer will want to arrive at a cinema before a movie's specific start time and stay at the cinema as long as the movie is scheduled to last. Given all these input parameters, our objective is to minimum the travel time on the road networks. In this problem, the planned stay period at any location is not considered as part of total travel time. What interested us is the travel time on the road.

Problem: The BESTMTNN Query
Given:

- A query point
- Distance metric - road distance
- k different types of points of interest
- Spatial-temporal road networks represented by Time Aggregated Encoded Path View (TAEPV) and Time Aggregated Multi-Type Graph (TAMTG)
- Time window constraints

Find:

- Least travel time, turn-by-turn route and BEst Start Time Multi-type Nearest Neighbor (BESTMTNN)

Objective:

- Minimize the travel time on the road networks from the query point covering an instance of each feature type and then back to the query point

3.3 BESTMTNN Algorithm

Here we examine in more detail that a typical BESTMTNN query from daily life that was introduced earlier. "Find a route with the best start time between 9:00 am and 11:00 am from my house through one Cub Food store (stay for about 1 1/2 hours), one Best Buy store (stay for about 1 hour) and one post office (arrive before 4:00 pm, stay about half an hour) and return to my house before 8:00 pm." From this typical query, we can find some important properties that could be used to guide the algorithm design for a BESTMTNN query.

First, normally a traveler queries the best start time within a defined time window (e.g., "between 9:00 am and 11:00 am"). By contrast, he's willing to return home at any time as long as it's before his latest time (e.g., "Return to my house before 8:00 pm"). We can see therefore that the window containing possible start times is much smaller than the "start - return" time window. This suggests that a forward search, beginning from the starting query point, may be preferred for most cases.

Second, the query starts from a query point (e.g., "my house") and ends at the same query point (e.g., "return to my house"). This basically says that the traveler asks for a closed travel route, which is different from the queries studied in previous works [36, 40, 54, 55]. In those studies, the requested travel routes do not include routes returning to the query point. This means that some properties identified for designing previous algorithms may not always hold. More specifically, the property 2, which guarantees the correctness of the LORD and RLORD algorithms in [54], is not always correct, and thus new properties should be identified in the design of algorithms for closed route queries.

Third, the traveler asks for a specific turn-by-turn route. Here we can differentiate the two levels of routes. The first level is the route through only the points of interest (POIs) without routing and scheduling on points that are not in the set of POIs. The second level of route is the route between two POIs. There may be multiple routes between these two points on road networks so there is a routing and scheduling issue. The BESTMTNN algorithm and corresponding data structure used in the algorithm should support both levels of routing and scheduling.

Finally, a BESTMTNN query is a temporal query; embedded in the query's search for the best start time and route is the assumption that traffic patterns change over time. There are three kinds of temporal patterns: long-term trends used in long-term forecasts, short-term information available when starting travel and used in short-term forecasts, and dynamic perturbation available only when arriving at a destination, and representing any unforeseen events encountered during travel. It is known that traffic volumes exhibit typical long-term temporal patterns. That is, traffic volumes vary at different time points known in advance. This kind of information can be used to do long-term forecasts. Due to the complexity of BESTMTNN queries, we do not consider short-term forecasts or dynamic perturbation in this chapter.

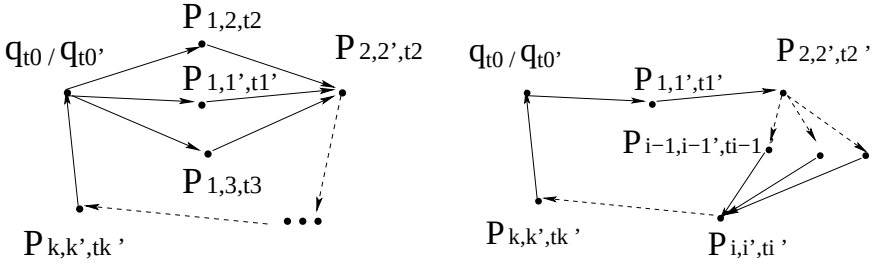
As stated above, the BESTMTNN query differs from previously studied queries and thus new properties need to be identified and existing data structures extended to support solutions to this query.

3.3.1 BESTMTNN-Related Properties

The BESTMTNN algorithm enumerates all permutations of all feature types. For each permutation, it starts with partial route only containing the query point and grows a partial route by adding points from next feature to a current partial route.

In this part, we discuss some BESTMTNN-related properties that guarantee the corretness of partial route growth procedure.

Property 1 If a route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2}, \ldots, P_{k,k',tk}, q_{t0'})$ is the optimal route, then the travel time of route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2}) = t(R(q0, P_{1,1',t1})) + t(R(P_{1,1',t1}, P_{2,2',t2}))$, is shortest among all possible routes from the query point q through any point from feature type 1 at any time and then reaches point $P_{2'}$ from feature type 2 at time $t2$.



(a) Property 1 (b) Property 2

Figure 3.1: Properties related to BESTMTNN query

In this property statement $P_{i,i',ti}$ represents point $P_{i'}$ of feature type i starting at time ti if the point $P_{i'}$ is not an end point on a route. If the point $P_{i'}$ is an end point on the route, $P_{i,i',ti}$ represents point $P_{i'}$ of feature type i with arrival time ti. Figure 3.1 (a) illustrates property 1. In this figure, the route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2})$ from the query point q_{t0}, through point $P_{1,1',t1}$ from feature type 1 and reaching point $P_{2,2',t2}$ from feature type 2 has the least travel time. However, when growing a partial route from the query point q to feature type 1, we don't know which specific partial route $R(q_{t0}, P_{1,1i,t1i})$ will lead to a shortest path from q_{t0} and reaching $P_{2,2',t2}$. Therefore, in order to grow partial routes from the query point q through a feature type 1 point to a feature type 2 point, we need to store the least travel time from the query point q through all feature type 1 points for all qualified time points.

This property is important and necessary because the BESTMTNN query route is a closed route that starts and ends with the same query point. Based on this property, we know that beginning a BESTMTNN query requires storing all the partial routes from the query point to all the points in the first feature type for all time points. This property also makes it possible to use a forward search in order to find a closed route.

Property 2 If a partial route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2}, \ldots, P_{i-1,(i-1)',t(i-1)}, P_{i,i',ti})$ is part of an optimal route, then the travel time of this partial route is least among all partial routes starting with q and ending with $P_{i,i',ti}$ for specific arrival time point t_i.

In partial route calculations, it is possible to have multiple partial routes ending with the same point at the same time point. Property 2 guarantees that only the partial route with the least travel time for a specific end point and time point needs to be stored. It also indicates that a stored partial route can be identified by its end point and arrival time point. According to property 2, it is enough to store information for the specific partial route $R(q_{t0}, P_{1,1',t1}, P_{2,2',t2}, \ldots, P_{i-1,(i-1)',t(i-1)}, P_{i,i',ti})$. Figure 3.1 (b) illustrates Property 2. In the figure, the partial routes ending with points from feature type $i-1$ are either from a different location or from a different time point. Before growing the partial route to feature type i, it is unknown which partial route ending with a point from feature type $i-1$ will be on the optimal route. However, according to property 2, it is known that the partial route must have a least cost (travel time) if it is on the optimal route. Therefore, in order to grow partial routes from feature type $i-1$ to feature type i it is enough to store those least cost partial routes ending with different points from feature type $i-1$ on different time points. After growing from all these partial routes to a specific point $P_{i,i',ti'}$ of feature type i at specific time ti' there are still multiple partial routes. All these newly grown partial routes could be represented as $R(q_{t0}, \ldots, P_{i-1,(i-1)',t(i-1)'}, P_{i,i',ti'})$, $R(q_{t0}, \ldots, P_{i-1,(i-1)'',t(i-1)''}, P_{i,i',ti'})$ etc. At this time, we know if the partial route ending with $P_{i,i',ti'}$ is on an optimal route it must be shortest. So, we only need to store one partial route ending with $P_{i,i',ti'}$ that has least cost.

3.3.2 Time Aggregate Multi-Type Graph (TAMTG)

A TAEPV stores all pairs of least travel times among all points, the start time of the route with the least travel time, and the next hop on the route at all time points on a graph. However, not all of this information is needed for partial route growth in a BESTMTNN query. Of interest in partial route growth are the least travel time among the points of interest (POIs) from different feature types at all time points and the start time of the route with the least travel time. In other words, there is no need to store the least travel time involved in a point of non-interest or if the least travel time is among the POIs from the same feature type. We call the graph that captures this more relevant set of least travel times a Time Aggregate Multi-Type

Graph (TAMTG).

As an illustration, the "initial" part of the Figure 3.2 shows part of TAMTG. Indicated on the graph is the least travel time from the query point q to points $r1$ and $r2$ from feature type r for time points 2, 3 and 4 and the least travel time from points $r1$ and $r2$ to point $b1$ from feature type b for time points 4 to 14. It is worth noting that there may not be a direct link between points. Instead in the TAMTG graph, the least travel time between two points has been calculated in advance. In addition, for simplicity the graph does not show all the calculated least travel times among points. For example, the least travel time from $b1$ to $r1$ and $r2$ is not given.

3.3.3 Partial Route Growth

A point on a partial route contains not only the point itself but also a specific time point. For example, $P_{i,j,tj}$ represents the point P_j on the partial route from feature type i at time point tj. Both spatial and temporal data are required because traffic volume varies over time, that is, the time dimension plays an important role in the modeling of spatial-temporal road networks. The current partial route may come from different previous partial routes for different time points. In other words, to answer a BESTMTNN query it is not enough to store location (point) information. What is needed is point information for all qualified time points. Due to the time window constraints, not all time points may qualify when searching for a best route. As discussed in the BESTMTNN properties section, a partial route can be identified exclusively by its end point and the arrival time at the end point because only partial routes with specific end points and specific time points are needed to grow the partial route to the next feature type. In summary, identifying a partial route requires storing a location and a time point along with a least travel time.

In daily life, when traffic volume is very high, drivers often choose to stay at their location instead of trying to drive at that moment. Common sense tells them that driving during high volume traffic will not expedite their travel or will do so just a little. If staying extra time at a location leads to shorter or equal travel time later, it is reasonable to choose staying extra time units at a location. In partial route calculation of the BESTMTNN query algorithm, it is possible to add extra time units to a location, but this extra time needs to be counted as part of travel time since it is not a planned stay. Therefore, there are two categories of cost at a partial route. One is directly generated from the growth of previous partial routes to the current

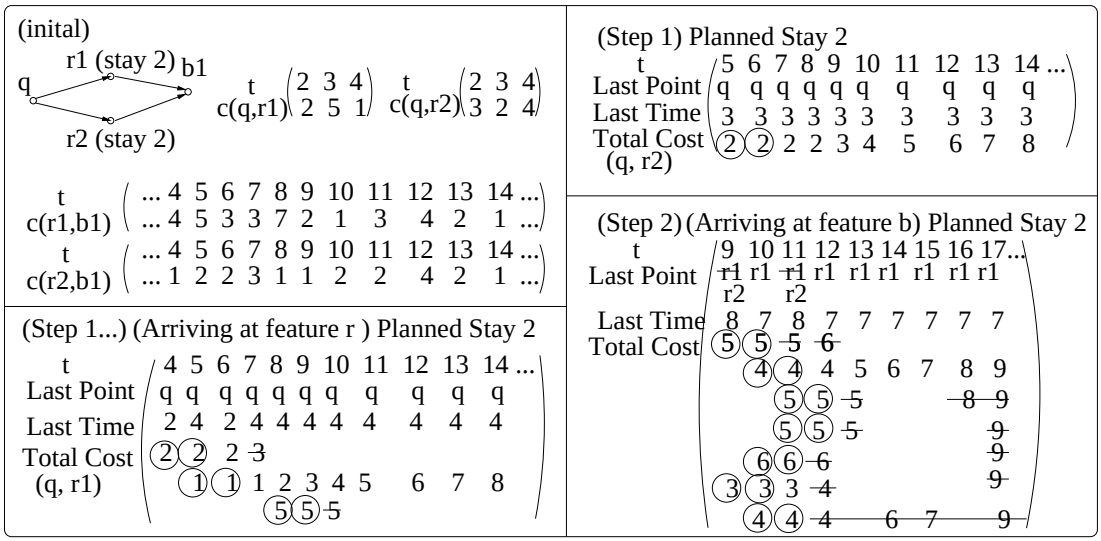


Figure 3.2: Partial route growth

partial route. In Figure 3.2, for example, the cost 4 of partial route $R(q_4, r_{1,7}, b_{1,10})$ is calculated from the growth of partial route $R(q_4, r_{1,7})$. Since there is a planned stay of 2 units at feature type b, the partial routes $R(q_4, r_{1,7}, b_{1,10})$ and $R(q_4, r_{1,7}, b_{1,11})$ cannot be grown at times 10 and 11. The number 4 inside a circle in "Step 1" and "'Step 2" indicates that this cost cannot be used for next partial route growth. For the same 2-unit planned stay, the cost of partial route $R(q_4, r_{1,7}, b_{1,12})$ is kept as 4. The other cost arises from any extra stay on the partial route that ended with the same point. For example, the cost 5 of partial route $R(q_4, r_{1,7}, b_{1,13})$ results from 1 extra time unit stay at point $b1$ at time 12.

When updating an existing partial route or generating a new partial route, it is necessary to update or generate the route for all qualified time points from all previous partial routes. The implication is again that a partial route in a BESTMTNN query is identified by both the end point on the route and a qualified time point. Here a qualified time point means a time point within the query time window for a currently visited feature.

Figure 3.2 illustrates the partial route calculation procedure. In the figure, t represents a time point. *Last Point* stores the last point on the partial route before reaching the current feature type. *Last Time* stores the start travel time from *Last Point* to the current point. *Total Cost* stores the least travel time for this partial route so far. The *Total Cost* number inside the circle means the time point with this cost is still within the planned stay period. So, this cost cannot be used to grow the

partial route to next feature. For example, if a traveler arrived at a Best Buy store at 3:00 pm with a cost 9 and planned to stay 1 hour, the traveler will not depart within the period from 3:00 pm to 4:00 pm. In our algorithm, no time point within this window can be used to grow the partial route. If all costs at a time point calculated from different previous partial routes are within the stay period, the traveler cannot depart from this time point. In other words, it is impossible to grow a partial route from this time point. A number with a crossed out line means this cost is larger than or equal to the previously calculated cost from a different previous partial route. Due to space limitations, we don't display all the crossed out *Last Point* and *Last Time*. In real computation, if updating a cost for a partial route, the corresponding *Last Point* and *Last Time* also need to be updated.

As shown in the example of Figure 3.2, we are searching the BESTMTNN for the feature type sequence $< r, b >$. Point q is the query point. There are two points r_1 and r_2 from feature type r and one point $b1$ from feature type b. The planned stay at a point of feature type r is time unit 2 and will not be counted as travel cost. The query is asking for the best start time between 2 and 4. The TAMTG shown in the initial step stores the least travel time of point pairs for all time points for the sequence $< q, r >$ from query point q to feature type r and feature type sequence $< r, b >$ from feature type r to feature type b. Step 1 shows growing the partial route from the route containing only the query point q to feature type r. Step 2 illustrates growing the partial route from the route of $< q, r >$ to feature type b. The rule for growing the partial route in both steps is the same. The following example shows how partial route $R(q_4, r_{1,7})$ is grown to become partial route $R(q_4, r_{1,7}, b_{1,10})$. More specifically, we explain how the total cost 4 is calculated on the partial route $R(q_4, r_{1,7}, b_{1,10})$ that is for the route $R(q, r1, b1)$ at time point 10 in step 2. In step 1, after staying for 2 time units at time point 5 at point $r1$ on partial route $R(q_2, r_{1,5})$, time point 5 becomes 7. Please note that time unit 2 is a stay planned in advance so it is not counted as part of travel cost. Then we find the cost (least travel time) from $r1$ at time point 7 to $b1$ in the initial setup, which is 3. The arrival time point at $b1$ is $7 + 3 = 10$ and the total cost is $1 + 3 = 4$. So the total cost of arriving $b1$ at time point 10 is 4. The partial route for this calculation is $R(q_4, b_{1,7}, r_{1,10})$. The cost 5 in the same row at time 13 is the cost after staying at $b1$ for one extra time unit. This calculation continues for all possible partial routes. For a specific point at a specific time point, the least travel time is stored as the *Total Cost* (least travel time) for

this partial route.

3.3.4 BESTMTNN Algorithm

Figure 3.3 illustrates the BESTMTNN algorithm. Briefly, the algorithm proceeds by gradually growing partial routes until finally a complete closed route is found. According to our analysis of this query's properties, a forward search strategy is preferred. In the following description of the BESTMTNN algorithm, assume a search order defined by $< F_1, F_2, F_3, \ldots, F_k >$. F_i represents feature type i.

In the priority Q, every node is attributed with a time series among which the i^{th} entry represents the partial route ending with the node at time point i and containing information about *Last Point*, *Last Time*, and *Total Cost* for this partial route as discussed in section 3.3.3. Thus a node in Q represents partial routes that end with the node for all qualified time points. For example, node u in the queue Q represents all the partial routes from query point q to u. When visiting a new feature type, a node from the new feature is added to the currently examined partial route to form a new partial route that is then enqueued into priority queue Q at the end for all time points within the time window of u. When visiting a feature type that has already been visited, the algorithm uses a label correcting approach [9] to modify the entries in a node according to the following conditions:

$C_u[t_{i-1} + \sigma_{vu}[t_{i-1}]] = \text{minimum}(\ \sigma_{vu}[t_{i-1}] + C_v[t_{i-1}],$

$C_u[t_{i-1} + \sigma_{vu}[t_{i-1}]])$ where

$C_u[t]$:Least travel time of partial route from query point to u arriving at time t

$\sigma_{vu}[t]$:Least travel time from v to u starting at time t

The algorithm maintains a list of partial routes in the priority queue Q. The priority query is ordered by the *Minimum Total Cost* of all partial routes that end with the same node at all time points. The *Total Cost* of the partial route at a time point is the least travel time spent on the partial route at a time point. Then a *Minimum Total Cost* is the minimum of *Total Cost* at all time points. After a new partial route is formed or an existing partial route is updated the partial route can be moved forward if its *Minimum Total Cost* is smaller than that of the prior partial route in the queue. This condition guarantees that the following partial routes in the queue cannot have smaller least travel times even if these partial routes could be updated from prior partial routes in the queue. For example, assume the queue contains partial routes $< R(q, r2)$, $R(q, r1, b1)$, $R(q, r1, b2) >$ ordered by *Minimum Total Cost*.

Algorithm BESTMTNN(q, TW, k, TAMTG)
Input : Query point q, Time Window Constraints (TW), number of feature types k , Distance metrics, Time Aggregated Multi-Type Graph (TAMTG), $\sigma_{vu}(t)$ - cost from v to u at time t
Output : BESTMTNN route

```
1.  Initialize : Add two fake new features of q as first (feature 0) and
2.  last feature (feature k + 1)
3.  Find greedy route and get Current Search Bound
4.  While there is permutation left
5.      Clear Q and enqueue q into Q with cost 0
6.      While priority Queue Q not empty
7.          v = Dequeue(Q)
8.          if (v is q and q is back-home query point OR
9.          Minimum Total Cost >= Current Search Bound)
10.             Search in next permutation
11.         i = NextFeature(v)
12.         for (each node u in feature i)
13.             for (every entry t_{i-1} within time window of feature i - 1)
14.             if (WithinTW(t_{i-1} + σ_vu[t_{i-1}], TW) AND
15.             ((C_u[t_{i-1} + σ_vu[t_{i-1}]] > σ_vu[t_{i-1}]+ C_v[t_{i-1}] OR i == 1))
16.                 C_u[t_{i-1} + σ_vu[t_{i-1}] = σ_vu[t_{i-1}] + C_v[t_{i-1}]
17.                 Update related information
18.             if (i has not been visited AND
19.             C_u[t_{i-1} + σ_vu[t_{i-1}]]+ σ_uq[t_{i-1} + σ_vu[t_{i-1}]) >
20.             Current Search Bound
21.                 Enqueue(u, Q)
22.             Maintain priority queue Q by moving u forward in Q
23.             according to Minimum Total Cost comparisons
24. Report current route as BESTMTNN route and the starting time of
25. BESTMTNN route as best starting time
```

Figure 3.3: BESTMTNN algorithm

(For simplicity, time tag has been ignored.) A new partial route $R(q, r2, b1)$ is grown from the partial route $R(q, r2)$. The partial route $R(q, r1, b1)$ could be updated to $R(q, r2, b1)$ if $R(q, r2, b1)$ has smaller *Minimum Total Cost*. However, the *Minimum Total Cost* of the newly updated partial route $R(q, r2, b1)$ would still be bigger than that of $R(q, r2)$. So, if the *Minimum Total Cost* of $R(q, r2)$ is bigger than *Current Search Bound* or the length of time series it is safe to stop the search in the current permutation.

More specifically, the first step of the algorithm after initialization is to find a greedy route quickly and use its cost as first *Current Search Bound*. In a greedy route search, first a random point of feature type F_1 is picked, then the cost of travelling from query point q at the first qualified time point to this point is used as the current *Total Cost*. Then, a random point from feature type F_2 is picked to grow the partial route. This procedure continues until the search returns to the query point. It is possible that this approach cannot find a qualified greedy route. In this case, the length of the time series is used as the *Current Search Bound*.

The next step keeps all qualified partial routes $R(q_{tj},\ P_{1,i,ti})$ as the first partial route set and enqueue all the partial routes into a priority queue Q that is ordered by *minimum Total Cost* of $C(q_{tj},\ P_{1,i,ti})$. Here the point $P_{1,i,ti}$ is any point P_i of feature type F_1 at specific time ti and $C(q_{tj},\ P_{1,i,ti})$ is the cost (least travel time) from q at time tj to point P_i of feature type 1 arriving at time ti.

In the following step the partial route at the head of priority queue Q is removed from the queue to become the current partial route. Assume this partial route is $R(q_{tj}, \ldots, P_{i-1,g})$ starting at time tj from q. On this partial route $P_{i-1,g}$ actually represents all partial routes ending with point $P_{i-1,g}$ for all qualified time points. If the next feature type F_i has not been visited, for every point $P_{i,l}$ in the feature type F_i, add the point $P_{i,l}$ to the current partial route, form a new partial routes $R(q_{tj}, \ldots, P_{i-1,g}, P_{i,l})$ and then calculate the new partial route costs $C(q_{tj}, \ldots, P_{i-1,g,tg}$ $P_{i,l,tl})$ for all qualified time points among which $P_{i,l,tl}$ is the point P_l from the feature type F_i at time point tl. Finally the algorithm finds the *Minimum Total Cost* as minimum$(C(q_{tj},\ \ldots,\ P_{i-1,g,tg}\ P_{i,l,tl}))$ for all qualified time points and enqueues the new partial route if the *Minimum Total Cost* is less than *Current Search Bound*. If the next feature type F_i has already been visited, there must be another partial route ending with $P_{i,l}$ in the queue Q. Look for this partial route in Q and compare the new least travel time on the new partial route to the previously calculated least

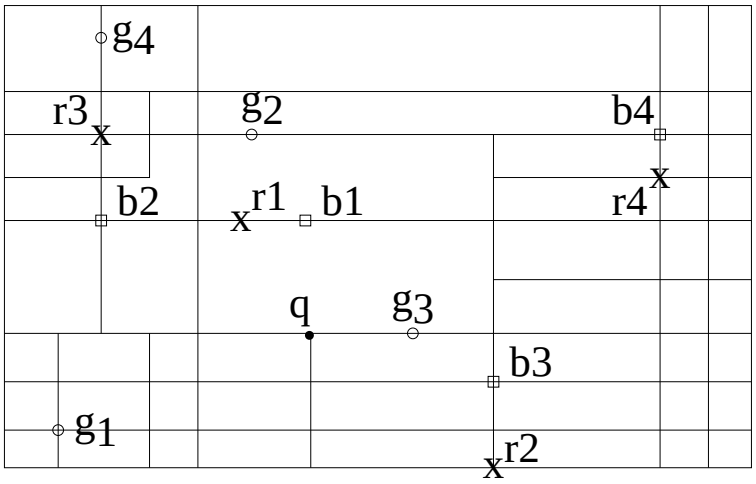


Figure 3.4: An example of BESTMTNN

travel time of the partial route ending with the same point $P_{i,l}$ for every qualified time point. If the new least travel time is less than the previous one at a time point, replace the previous partial route with the new partial route for this time point. Similar replacements should be done for all qualified time points.

In the last step in the iteration, the algorithm moves the partial route ending with the point $P_{i,l}$ forward in Q to keep the priority queue Q sorted by *Minimum Total Cost*.

This procedure will continue until a complete closed route is found for this permutation or the *Minimum Total Cost* from the current examined partial route is greater than the *Current Search Bound* or the length of the time series. At this time, it is possible that some partial routes remain in the priority queue. However, since the queue is sorted by *Minimum Total Cost*, it is impossible to find another complete closed route from the partial routes remaining in the queue with less travel time than the *Current Search Bound* or the length of the time series.

After searching all permutations, the BESTMTNN algorithm generates a complete closed route consisting of POIs with the best start time and a shortest travel time. The full turn-by-turn route can be found by simply checking with TAEPV that is used to generate TAMTG.

3.3.5 An Example of BESTMTNN Algorithm

Figure 3.4 illustrates how the BESTMTNN algorithm works on a spatial-temporal road network. For simplicity, we only show the algorithm for a specific permutation

in the following. The full BESTMTNN algorithm works without pre-defined search order. In this example, q is the query point and there are three feature types r, b and g. Assume the current search sequence is $< r, b, g >$. First, the query point q is enqueued and then dequeued to calculate partial routes $R(q, r_1)$, $R(q, r_2)$, $R(q, r_3)$ and $R(q, r_4)$ for all qualified time points as described in section 3.3. (For simplicity, the time dimension of partial routes is not shown in this example.) $R(q, r_1)$ or $R(r_1)$ represents all partial routes ending with point r_1 for all time points. These partial routes are enqueued and sorted by *Minimum Total Cost* from all partial routes. Assume now the sorted partial routes in the priority queue are $< R(r_1), R(r_2), R(r_3), R(r_4) >$. Next, partial route $R(r_1)$, that is $R(q, r_1)$, is dequeued and grown by adding every point in feature type b to it. Four new partial routes $R(q, r_1, b_1)$, $R(q, r_1, b_2)$, $R(q, r_1, b_3)$ and $R(q, r_1, b_4)$ are generated and inserted into the queue such that the queue remains sorted. Assume the queue is $< R(r_2), R(b_1), R(b_2), R(r_3), R(r_4), R(b_4), R(b_3) >$ after sorting. At this time $R(r_2)$, that is $R(q, r_2)$, is dequeued and four new partial routes $R(q, r_2, b_1)$, $R(q, r_2, b_2)$, $R(q, r_2, b_3)$ and $R(q, r_2, b_4)$ are generated by adding every point in feature type b to it. Since feature type b was visited, every new partial route identified by its end point and arrival time at the end point is compared to the existing partial route if both of them share the same end point and time point. The partial route with shorter *Total Cost* is kept. After all the partial routes are grown and sorted, the partial route ending with point b_3 is moved forwarded to the head of the priority queue and the priority queue becomes $< R(b_3), R(b_1), R(b_2), R(r_3), R(r_4), R(b_4) >$. This procedure continues until a complete closed route ending with the query point q becomes the head of the priority queue. The BESTMTNN for this permutation is the complete closed route with the smallest least travel time value among all complete closed routes. In this example, the route with *Minimum Total Cost* among all routes represented by $R(q, r_2, b_3, g_3)$ is the BESTMTNN route.

3.4 Experimental Evaluations

In this section, we present the results of various experiments to evaluate whether our BESTMTNN algorithm is optimal in finding the shortest path in terms of least travel time for the BESTMTNN query in real road network data sets. Specifically, we demonstrate performance of the BESTMTNN algorithm with respect to execution time using road network data with different properties related to the number of feature

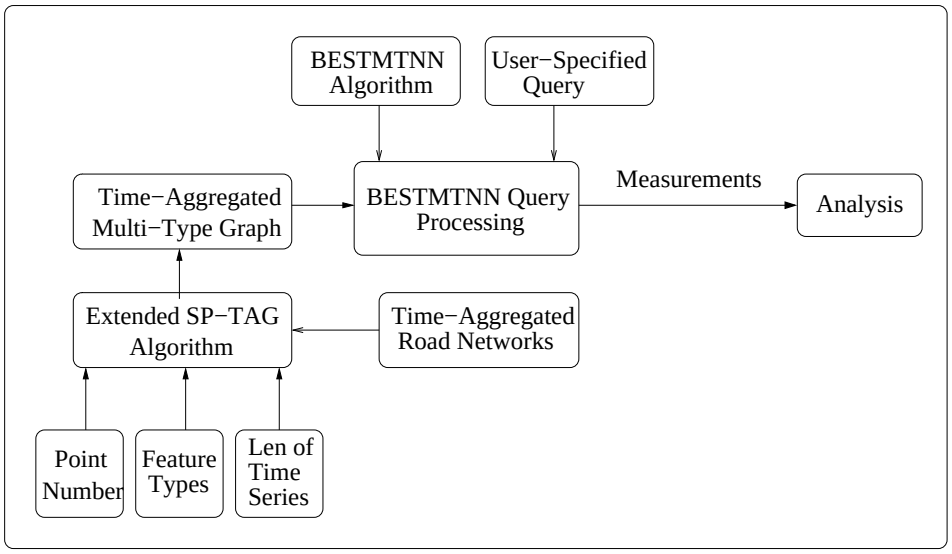


Figure 3.5: Experiment setup and design

types, number of points in each feature type, length of query windows and length of time series.

3.4.1 The Experimental Setup

Experiment Platform Our experiments were performed on a PC with two 2.83 GHz CPUs and 4 GByte memory running the GNU/Linux Ubuntu 8.04.2 operating system. All algorithms were implemented in the C programming language.

Experimental Data Sets We evaluated the performance of the BESTMTNN algorithm for the BESTMTNN query with real road network data. The data set represents the static digital road map from the area of 3 miles in downtown Minneapolis, Minnesota. In this data there are a total of 786 nodes and 2106 edges. We synthetically generated different lengths of travel time series to create different Time-aggregated Road Networks. For evaluation purposes, data points were randomly picked from the nodes on the road networks to represent the points of interest (POIs) from different feature types.

There were four different parameters in our experimental setup.

- Feature Type (FT): Feature type numbers from 2 to 7 to show the scalability of the algorithm in terms of number of feature types.
- Number of Points (NOP): Number of data points from 20 to 120 in each feature type.

- Length of Time Windows (TW): Length of query time windows from 20 to 120 for each feature type.
- Length of Time Series (TS): The available number of time points from 50 to 300 representing the long-term traffic patterns.

Experiment Design Figure 3.5 describes the experimental setup to evaluate the impact of design decisions on the relative performance of the BESTMTNN algorithm for the BESTMTNN query. The SP-TAG algorithm described in [22] calculated the shortest path in terms of least travel time between a pair of spatial points for a given start time. An extended version of SP-TAG algorithm randomly picks up points from the Time-Aggregated road networks as POIs in different feature types and generates TAMTG with different parameters choices related to number of feature types, number of points in a data set belonging to a feature type and different lengths of time series, based on the Time-Aggregated Road Networks. The BESTMTNN query processing engine takes different TAMTGs and user-specified queries to generate the performance measurements that are analyzed to evaluate the performance of the BESTMTNN algorithm. The user-specified query gives the start time interval during which the algorithm will find the best start time, the length of query time windows defining the qualified visit time interval at all feature types, stay time intervals at all feature types and the returning home time interval.

Our goal was to answer the following questions: (1) How do changes in number of feature types affect the scalability of the BESTMTNN algorithm? (2) How do differences in number of data points in each feature type affect the performance of the algorithm? (3) How do the lengths of query time windows at each feature type affect the performance of the algorithm? (4) How do the lengths of time series (number of time points) affect the performance of the algorithm?

3.4.2 Scalability of BESTMTNN with Respect to Feature Types

This section describes the scalability of BESTMTNN in terms of the number of feature types. In this experiment, the number of points in each feature type is 40, the length of query time windows for all feature types is 60, and the length of time series is 300 time units. The number of feature types changes from 2 to 7. Figure 3.6 shows that the algorithm runs less than 0.1 seconds when feature type number is 2,3 and 4 and runs for less than .5 seconds at feature type number 5. Run time increases to 3

seconds with 6 feature types and to a little less than 20 seconds for 7 feature types. The execution time increases dramatically when the number of feature types increases from 5 to 7 because the number of permutations increases dramatically, that is, the number of iterations required for searching BESTMTNN becomes significantly large. These results show that BESTMTNN is scalable with up to 7 feature types and for most daily life queries, BESTMTNN can give the query results quickly.

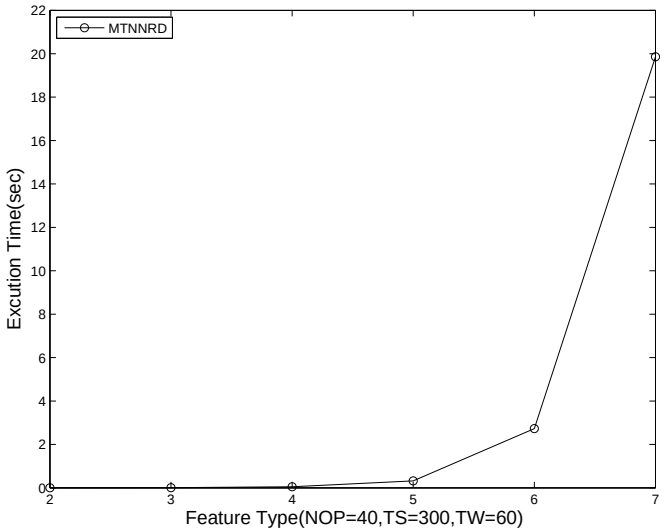


Figure 3.6: Scalability in terms of feature type

3.4.3 The Effect of Number of Points in Feature Types on The Performance

In this section, we show how the POI density of the data sets affects the performance of the BESTMTNN algorithm. We tested BESTMTNN with feature type number 7, length of query time windows 60, time series units 300 and a changing number of points in each feature type from 20 to 120. The results shown in Figure 3.7 indicates the data density in the area covered by the experiment data set affects the BESTMTNN performance in a near linear fashion. When the data point number is 20, the running time is about 6 seconds. However, when the data point number reaches 120, that is, the total number of POI is $120 \times 7 = 840$, the running time is about 80 seconds. In our road network data set, the total node (point) number is 786, which indicates that there must be some POIs from different feature types that share the same location. This is a reasonable situation in daily life. For example, multiple business units may share the same building or mall. Meanwhile, a POI total

of 840 means the data is extremely dense. It is probably necessary to partition this dense area into smaller areas in order to answer a BESTMTNN query faster.

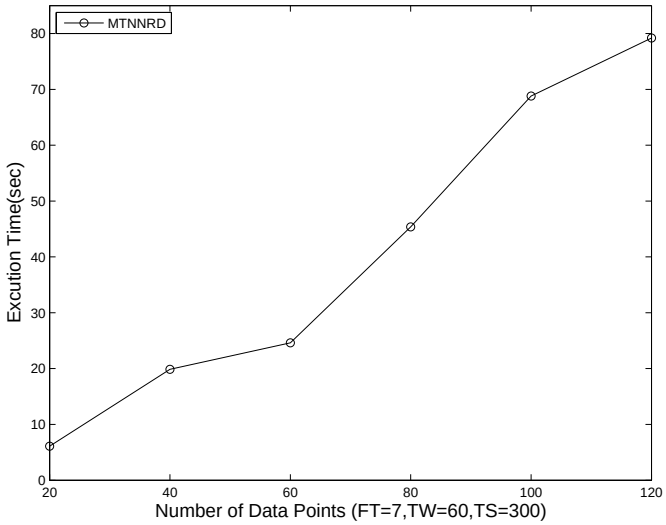


Figure 3.7: Effect of number of points

3.4.4 The Effect of Different Lengths of Query Time Windows on Performance

In this section, we illustrate the BESTMTNN performance under different lengths of query time windows at each feature type. We set the feature type number at 7, the number of points in each feature type at 40 and the length of time series at 300. We took the same query windows size for all feature types and changed the lengths of query time windows from 20 to 120 units. Figure 3.8 shows that the BESTMTNN query is sensitive to the length of query window. Run times increase near linearly with increases of query time window. When the length of the time window is 20, the running time is less than 2 seconds. When the time window reaches at 120, the running time is just shy of 40 seconds. These results tell that it is beneficial for users to specify smaller query time windows. However, it is worth remembering that the chance of getting a BESTMTNN decreases as the size of the query time window becomes smaller.

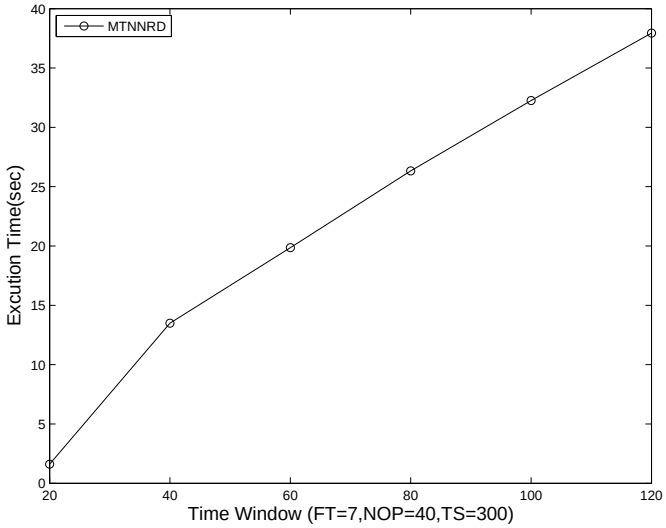


Figure 3.8: Performance under different time window sizes

3.4.5 Effect of Different Lengths of Time Series on Performance

This experiment evaluates the effect of the total number of time series units in the spatial-temporal road network on the performance of BESTMTNN. Here we used number of feature types 7, number of points in each feature type 40 and the length of query time windows 50 or 60. The length of the time series was changed from 50 to 300. Please note the maximum meaningful length of query time window is 50 when the total available time series unit is 50. From the Figure 3.9, we can see that the total running times are between 16 and 21 seconds. The lengths of time series affect the BESTMTNN performance only a little and the changing pattern is not significant when the length of time series changes.

3.5 Summary

We identified the properties of a BESTMTNN query and formalized a BESTMTNN query problem on spatial-temporal road networks. We extended the EPV from spatial only to spatial-temporal road networks and utilized a special case of the extended EPV (TAEPV), TAMTG, in designing our BESTMTNN algorithm based on the label-correcting approach. In our experiment we evaluated the performance of the BESTMTNN algorithm in terms of number of feature types, number of points in each feature type, the length of query time windows and the length of time series of road networks.

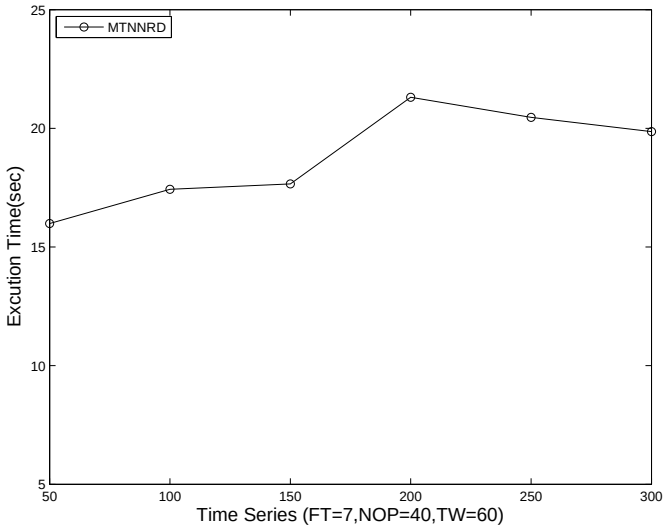


Figure 3.9: Effect of time series length

In the future work, we plan to conduct the comparative experiments to characterize dominance zones of alternative choices for critical algorithm design decisions and extend the BESTMTNN algorithm to huge road networks by using a hierarchical TAMTG since it is extremely time-consuming to answer BESTMTNN query with current technologies without partitioning huge road networks.

Chapter 4

Multi-Type Reverse Nearest Neighbor Search

This chapter presents a study of the Multi-Type Reverse Nearest Neighbor (MTRNN) query problem. Traditionally, a reverse nearest neighbor (RNN) query finds all the objects that have the query point as their nearest neighbor. In contrast, an MTRNN query finds all the objects that have the query point in their multi-type nearest neighbors. Existing RNN queries find an influence set by considering only one feature type. However, the influence from multiple feature types is often critical for strategic decision making in many business scenarios, such as site selection for a new shopping center. To that end, we first formalize the notion of the MTRNN query by considering the influence of multiple feature types. We also propose R-tree based algorithms to find the influence set for a given query point and multiple feature types. Finally, experimental results are provided to show the strength of the proposed algorithms as well as design decisions related to performance tuning.

4.1 Introduction

Given a data set P and a query point $f_{q,q}$, a reverse nearest neighbor (RNN) query finds all objects in P that have the query point $f_{q,q}$ as their nearest neighbor. When the data set P and the query point $f_{q,q}$ are of the same feature type, the RNN query is said to be monochromatic. When they are from two different feature types, the

query is bichromatic. An RNN is said to represent a set of points in P that has been influenced by a query point $f_{q,q}$ [31] and for this reason RNN queries have been widely used in Decision Support Systems, Profile-Based Marketing, etc. For example, before deciding to build a new supermarket, a company needs to know how many customers the supermarket may potentially attract. In other words, it needs to know the influence of opening a supermarket. An RNN query can be used to find all residential customers that live closer to the new supermarket than any other supermarket. If we assume that the presence alone of a new supermarket will determine whether customers choose to shop at it, results of this query may be used to decide whether to go ahead with the project. However, queries based on such assumptions may not always produce useful results. A decision to shop at the new store may also be influenced by the presence of other types of stores. For example, some customers may want the opportunity to shop for groceries, electronics, and wine. Here, what influences customers' choice of grocery store is the shortest route through one grocery store, one electronics store, and one wine shop rather than the shortest route to the grocery store alone. In this case, the RNN query needs to consider the influence of feature types besides grocery store. To date, both monochromatic and bichromatic RNN queries have considered the influence of only one feature type. As the above example shows, there is a need to also consider the influence of other feature types in addition to that of the given query point.

Figure 4.1 illustrates how two feature types affect the results of an RNN query. In the figure, point $f_{q,q}$ is the given query point and point $f_{q,1}$ is another point from the same feature type F_q, which we call the query feature type. Point $f_{1,1}$ is a point from a second feature type F_1. p_1 is a point from the data set P in which the RNNs will be found. The perpendicular bisector $\perp (f_{q,1}, f_{q,q})$ or l_1 between points $f_{q,1}$ and $f_{q,q}$ in Figure 4.1a divides the space into two half-spaces: $Plane(l_1, f_{q,1})$ containing point $f_{q,1}$ and $Plane(l_1, f_{q,q})$ containing point $f_{q,q}$. Any point falling inside half plane $Plane(l_1, f_{q,1})$ is closer to point $f_{q,1}$ than $f_{q,q}$. Similarly half plane $Plane(l_1, f_{q,q})$ contains all points that are closer to $f_{q,q}$ than $f_{q,1}$. For perpendicular bisector $\perp (f_{q,q}, f_{q,1})$ or l_2, these properties also hold.

In Figure 4.1a, if the influence of point $f_{1,1}$ is not considered, p_1 is an RNN of the given query point $f_{q,q}$ since $f_{q,q}$ is the nearest neighbor of p_1. However, when the influence of point $f_{1,1}$ is considered, p_1 is not an RNN of $f_{q,q}$ because the distance of the route $R(p_1, f_{q,1}, f_{1,1})$ is shorter than the distance of the route $R(p_1, f_{q,q}, f_{1,1})$. As

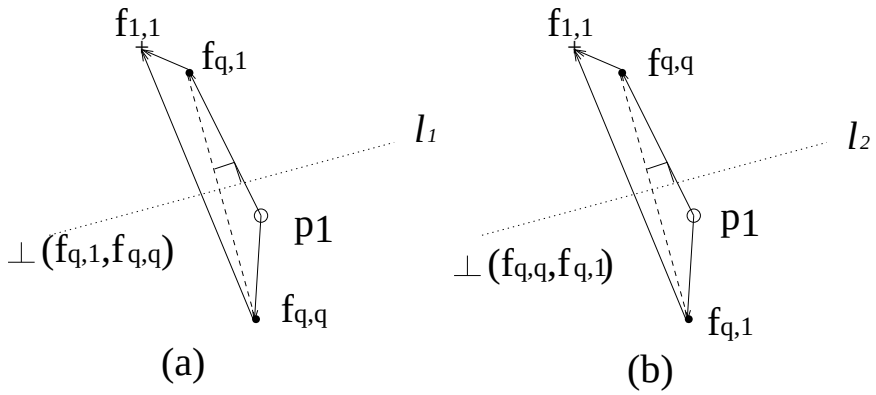


Figure 4.1: Influence of two feature types

an opposite example, in Figure 4.1b, when the influence of point $f_{1,1}$ is not considered, p_1 is not an RNN of the given query point $f_{q,q}$ since $f_{q,1}$ is the nearest neighbor of p_1. However, when the influence of point $f_{1,1}$ is considered, p_1 is an RNN of $f_{q,q}$ because the distance of the route $R(p_1, f_{q,q}, f_{1,1})$ is shorter than the distance of the route $R(p_1, f_{q,1}, f_{1,1})$. This example shows that an RNN query may give different results depending on the number of feature types accounted for. Implicit in the multi-type RNN search that we describe is a query to find the shortest distance from point p_1 through one instance of given feature types F_q and F_1. We call this query a multi-type nearest neighbor (MTNN) query in [40].

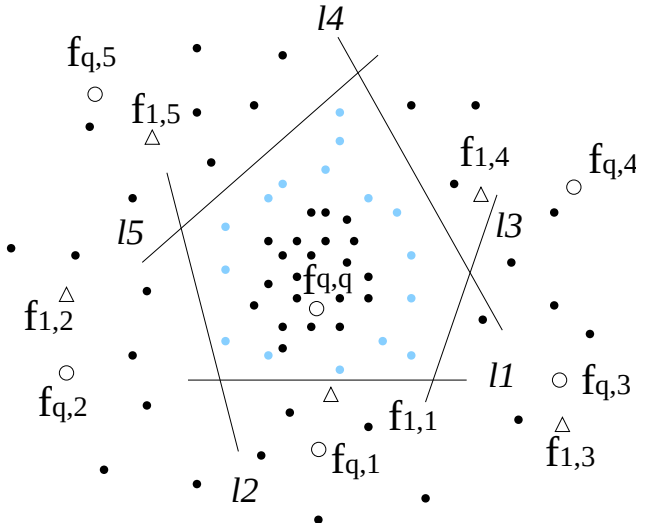


Figure 4.2: A use case

Figure 4.2 illustrates another business query problem. Assume that a business rule states "Build a new grocery store if the store will be a nearest neighbor of more

than 30 residences.". In the figure, assume that point $f_{q,i}$ is a grocery store from feature type F_q, grocery store. The new grocery store to be built by our grocery store chain is represented by the point $f_{q,q}$, which is the query point. Every grocery store is represented by a circle. Point $f_{1,i}$ is a wine shop from feature type F_1, wine shop. Every wine shop is represented by a triangle. The solid points represent residences. The lines l_1, l_2, l_3, l_4 and l_5 are perpendicular bisectors between point $f_{q,q}$ and $f_{q,1}$, $f_{q,q}$ and $f_{q,2}$, $f_{q,q}$ and $f_{q,3}$, $f_{q,q}$ and $f_{q,4}$, and $f_{q,q}$ and $f_{q,5}$ respectively. A classical RNN query considers the influence of the grocery store only and finds that all residences located inside the region formed by lines l_1, l_2, l_3, l_4 and l_5 have the point $f_{q,q}$ as their nearest neighbor. Thus the new grocery store $f_{q,q}$ is the nearest neighbor of these residences, and the residences are the reverse nearest neighbors of $f_{q,q}$. In this business context, shoppers from the residences are considered likely to visit the new grocery store $f_{q,q}$ because their route to the new store is shorter than to any other grocery store. Further, the number of residences totals 37 in this case, which is enough to justify going ahead with the project to build the new grocery store $f_{q,q}$.

However, this classical RNN query result does not account for shoppers who want the opportunity to buy not only groceries but also wine. (Some municipalities do not permit the sale of wine in grocery stores.) Such shoppers are interested in finding the shortest path through not one, but two types of stores, a grocery store and a wine shop. A reverse nearest neighbor search that considers the influence of the additional feature type wine shop may yield more useful results as shown in Figure 4.2. Indeed, after applying the same distance calculation method, i.e., the MTNN query defined in [40] and used previously in Figure 4.1, the shortest route from some of the residences (the blue or gray points in the figure) through one grocery store and one wine shop does not contain the new grocery store $f_{q,q}$. This means that these residences are no longer RNNs of the new store and not likely to visit it. More specifically, when two feature types F_q grocery store and F_1 wine shop are considered in the query, only 21 residences are found to be RNNs of $f_{q,q}$, which does not meet the threshold required for building a new store.

As this example demonstrates, an RNN query that considers the influence of more than one feature type can produce quite different results from an RNN query based on a single feature type. The differences can be two-fold. In our example above, the multi-type query returned a different number of RNNs. In other cases, the specific RNNs returned may differ. For example, a grocery chain may be interested

in knowing not only the number of potential shoppers but also the average household income of potential shoppers at a new grocery store. A query that considers the influence of the grocery store alone will generate one answer (e.g., average household income=$27,000), while a query that considers the influence of additional nearby businesses (e.g., wine shop and electronics store) may generate another (e.g. average household income = $42,000, reflecting the fact that a different set of customers is attracted to these shops). Whether it is a difference in the number of RNNs returned or the RNNs themselves, multi-type RNN queries have considerable potential to impact decision-making in business applications.

In this chapter we formalize the Multi-Type Reverse Nearest Neighbor (MTRNN) query problem to consider the influence of other feature types in addition to the feature type of the given query point. We propose R-tree based algorithms to prune the search space by filtering R-tree nodes and points that are definitely not the MTRNN of the given query point. Then refinement approaches are used to remove the false hit points. Our experiments on both synthetic and real data sets demonstrate that typical MTRNN queries can be answered by our algorithms within reasonable time. The design decisions related to performance tuning are provided as well. We also include experimental results that vividly highlight the degree to which MTRNN query results can differ from results of traditional RNN queries.

Related Work. The notion of a reverse nearest neighbor query was first formalized in a pioneering paper by Korn and Muthukrishnan [31]. They proposed algorithms for static and dynamic cases by pre-computing the nearest neighbor for each data point. Following their work, an on-line algorithm that utilized property of 2-dimentional space partitioning was proposed in [58] for dynamic databases and an index structure was devised to answer RNN queries by Yang and Lin in [17]. Since then RNN queries have been extended in a number of dimensions including metric space [3] and [62], high-dimensional space [56], ad-hoc space [73], large graphs [74] and spatial networks using Voronoi diagram [50]. RNN queries have further been studied in an extended family containing different variations of classic RNN problem.. Yao *et al.* [72] studied reverse furthest neighbors in spatial databases. Tao *et al.* [60] dealt with monochromatic RkNN queries in arbitrary dimensionality for the first time. Wu *et al.* [68] proposed a new algorithm for both monochromatic and bichromatic RkNN queries on 2-dimensional location data. Gao *et al.* [21] extended RNN to a visible RkNN problem, considering presence of obstacles and Vlachou *et al.* [65] proposed a reverse

top-k query problem to support the rank-aware query processing. Wong *et al.* [66] formalized a maximum bichromatic reverse nearest neighbor (MaxBRNN) problem that finds an optimal region maximizing the size of BRNNs. In this problem the location of the query point is not fixed, which makes it different from existing problems. RNN problem plays an important role in mobile computing. [28,69] researched continuous RNN and [64] studied bichromatic RNN in mobile system. Dellis and Seeger introduced the concept of Reverse Skyline Queries in [18] for the first time. Liang and Chen [37] then extended the skyline queries to uncertain databases. While RNNs have been studied for a large variety of computational problems, a common feature of all these studies is that they only consider the influence of a single feature type, the feature type of the given query point.

Where the effect of multiple feature types has been studied is in nearest neighbor queries. Sharifzadeh *et al.* [54] proposed an Optimal Sequenced Route (OSR) query problem and provided three optimal solutions, Dijkstra-based, LORD and R-LORD. Essentially, the OSR problem is a special case of the MTNN problem [40] because the order of feature types is fixed for the OSR problem. The OSR work was extended to road networks by using Voronoi diagrams in [55]. In this work, the visiting order of feature types is also fixed, which makes it different from the work in [38]. Recently Kanza *et al.* [35] formally defined an interactive route search problem in that the route is computed in steps and presented heuristic interactive algorithms for route-search queries in the presence of order constraints. In paper on Trip Planning Queries (TPQ) [36] that find a route through multiple features for all permutations of feature types, a number of fast approximate algorithms were proposed to give sub-optimal solutions. In the work [8] multiple spatial rules represented as a directed graph were imposed on sequenced route query and three algorithms were developed to find sub-optimal travel distance. These studies offer some insight into the handling of multiple feature types. However, if multi-type NN search approaches are to be extended to reverse nearest neighbor searches, they need to be able to handle non-fixed order visits and provide solutions finding optimal routes with shortest distance and then RNNs. Ma *et al.* [40] formalized a Multi-Type NN query problem and proposed a Page Level Upper Bound (PLUB) based algorithm to find an optimal route for the MTNN query without any predefined visiting order of feature types. This work was then extended to spatio-temporal road networks in [38]. In this chapter, we build on our study of multi-type reverse NN solutions with non-fixed visits. To our knowledge, ours

is the first to consider the influence of multiple features in reverse nearest neighbor problems.

Outline. The remainder of this chapter is organized as follows. Section 4.2 formalizes the MTRNN problem and presents a brute force algorithm as a baseline algorithm. In section 4.3 we propose two filtering methods to prune the search space and three refinement approaches to remove the false hit points. Section 4.4 analyzes the complexity of the algorithms and section 4.5 gives the experimental setup, results and a discussion. Finally, in section 4.6, we conclude and suggest future work.

4.2 Preliminaries

In this section, we introduce some basic concepts, explain some symbols used in the remainder of the chapter and give a formal statement of the MTRNN query problem. We also present a one-step brute force algorithm for the MTRNN query as a baseline algorithm that is directly based on the formulation of the MTRNN query problem. Table 4.1 summarizes the symbols to be used in the rest of this chapter.

Notation	Description
P	Queried data set
p_i	A point from queried data set P
F_q	Feature type q that the query point $f_{q,q}$ belongs to
F_i	Feature type i
$f_{q,q}$	A query point
$f_{i,j}$	A point j from feature type F_i
s_i	A point i in space
$s_i s_j$	Line segment $s_i s_j$
$len(s_i s_j)$	Length of line segment $s_i s_j$
$R(..., ...)$	A route through some given points
$d(R(..., ...))$	Distance of a route through some given points
R_i	An *R-tree index node*
fr	A *feature route*
S_{fr}	A set of *feature route*
S_{frps}	A set of *feature route point set*
S_c	A candidate MTRNN set

Table 4.1: Summary of Symbols

4.2.1 Problem Formulation

Let p_i represent any point in the queried data set P, $\{p_1, p_2, \ldots, p_n\}$ be a point set, F_i be a feature type, and $f_{i,i'}$ be a point $f_{i'}$ from feature type F_i.

Definition 1 Partial route. *A Partial Route $R(p_i,\ f_{1,1'},\ f_{2,2'},\ ...,\ f_{l,l'})$ is a route from point p_i through points from different feature types $F_1, F_2, ..., F_l$.*

$d(R(p_i,\ f_{1,1'},\ f_{2,2'},\ ...,\ f_{l,l'}))$ is the distance of the partial route.

Definition 2 Multi-Type Nearest Neighbor (MTNN). *An MTNN of a given point p_i is defined to be the ordered point sequence $< f_{1,1'}, f_{2,2'}, \ldots, f_{k,k'} >$ such that $d(R(p_i, f_{1,1'},\ f_{2,2'}, ..., f_{k,k'}))$ is minimum among all possible routes from p_i through one instance of feature types $F_1, F_2, ..., F_k$.*

$R(p_i, f_{1,1'},\ f_{2,2'}, ..., f_{k,k'})$ is called the MTNN route for the given point p_i. Please note although an MTRNN route resulted from an MTRNN query is an ordered sequence, the order of feature types is not specified in an MTNN query.

Definition 3 Multi-Type Reverse Nearest Neighbor (MTRNN). *An MTRNN of the given point $f_{q,q}$ is defined to be a point p_i in the queried data set P such that the MTNN of the point p_i contains the query point $f_{q,q}$.*

For a point p_i in set P and feature types F_1, ..., F_q, ... F_k, find the MTNN route $R(p_i, f_{1,1'}, f_{2,2'}, ..., f_{k,k'})$. If the given query point $f_{q,q}$ is on the MTNN route, the point p_i is an MTRNN of the query point $f_{q,q}$.

An MTRNN query is a query finding all MTRNNs for the given query point $f_{q,q}$, k feature types, and the queried data set P. As in the MTNN problem, the order of the given k feature types is not specified because an MTRNN query is trying to find the influence of given feature types. The features have influence no matter what order they are in. Therefore, we don't force any order constraint on the feature types for an MTRNN query.

In our problem formulation, we are querying against a data set P whose feature type differs from the given k feature types. We use Euclidean distance as the distance metric. The query point $f_{q,q}$ is from feature type F_q of the given feature types. Each data set of different feature types has its own separate R-tree index that will be used to find nearest neighbor in the algorithm Figure 4.4 and in the adapted MTNN

algorithm Figure 4.14 of the refinement step. The query finds all MTRNNs in the queried data set P in terms of k different feature types for the given query point $f_{q,q}$. That is, for every point in the queried data set P, first find its MTNN. If the given query point $f_{q,q}$ is on the MTNN of this query point, the queried point is considered to be an MTRNN of the given query point $f_{q,q}$.

The following is the formal definition of the MTRNN query problem.

Problem: The MTRNN Query
Given:

- A data set P to be queried against
- Distance metric: Euclidean distance
- k sets of points, each set coming from one of k different feature types
- A query point $f_{q,q}$, coming from feature F_q of the k feature types
- Seperate R-tree index for each data set including queried data set and feature data set

Find:

- Multi-type reverse nearest neighbors (MTRNNs) in the queried data set P for the given k different feature types and the query point $f_{q,q}$ such that $f_{q,q}$ is on the MTNN of each MTRNN point

Objective:

- Minimize the length of the route starting from an MTRNN covering the query point $f_{q,q}$ of F_q and one instance of every other feature type excluding feature type F_q

In our problem formulation, the objective function follows the definition used in [40, 8, 36, 54]. The objective function, however, in these studies is very general and does not consider the possible influence difference of different feature types. In other words, every feature type has the same influence on a point. It also does not consider the route returning to the query point, which may be useful in some applications. The objective function defined here incorporates these considerations. Thus, our objective function, represented as $d(R(p_i, f_{1,1'}, f_{2,2'}, ..., f_{k,k'}))$, is the distance of the route from

a point p_i through one instance of every feature of the given k feature types. If the shortest route found from point p_i contains the query point $f_{q,q}$, then this point p_i is an MTRNN of the query point $f_{q,q}$.

4.2.2 One Step Baseline Algorithm for the MTRNN Query

We developed a naive approach directly based on the concept of the MTRNN query problem and the relationship between MTNN and MTRNN queries. For the given query point $f_{q,q}$, this one-step algorithm simply scans all the points in the queried data set P. More specifically, for every point p_i in P, it finds the MTNN of this point in the given data sets of k different feature types using algorithm described in [54] for one permutation of feature types and then applying the algorithm for all permutations of feature types as did in [40]. If the query point $f_{q,q}$ is on the MTNN of a data point p_i, the data point p_i is an MTRNN of $f_{q,q}$. This brute force algorithm does not prune any data points so it is very time-consuming and not scalable. It is presented here as a baseline algorithm because it is directly based on the formulation of the MTRNN query problem and is useful for evaluating the correctness of our MTRNN algorithm.

The scalability of the MTRNN algorithm depends on the efficiency of the underlying MTNN search of all the points in the queried data set. Due to the complexity of the current MTNN algorithm [40, 54], achieving a scalable MTRNN algorithm requires designing an efficient filtering method to prune most of the queried data before the MTNN algorithm is applied. In the following section, we present an MTRNN algorithm with an efficient filtering step.

4.3 Multi-Type Reverse Nearest Neighbor Algorithms

Our MTRNN algorithm is an on-line algorithm consisting of three major steps, preparation, filtering, and refinement. The output of the MTRNN query that take into account the influence of multiple feature types is a set of reverse nearest neighbors. The preparation step in section 4.3.1 finds feature routes for the filtering step by applying a greedy algorithm that uses R-tree indexes from all feature types during searching. Thus, normally only a small portion of data will be examined. The filtering step in section 4.3.2 eliminates R-tree nodes that cannot contain an MTRNN by utilizing feature routes and then retrieves all remaining points that are potential MTRNNs to form a candidate MTRNN point set. In this section, we describe two

Algorithm MTRNN(R-trees, R-tree, $f_{q,q}$, F_q)
Input : R-trees for each feature, R-tree index root of queried data set, query point $f_{q,q}$, the query feature type F_q
Output : MTRNN set S_c

1. //1.Preparation Step
2. Partition Space and put subspace into set S_{sub}
3. //Find initial greedy route
4. S_{frps} = Greedy(R-trees, $f_{q,q}$, $\emptyset$)
5. Find feature route set S_{fr} from S_{frps}
6. S'_{fr} = FindFeatureRoutes(R-trees, $f_{q,q}$, S_{fr}, S_{sub})
7. $S_{fr} = S'_{fr} \cup \S_{fr}$
8. //2.Filtering Step
9. S_c = Filtering(R-trees,R-tree,S_{fr},$f_{q,q}$,NIL)
10. //3.Refinement Step
11. S_c = Refinement(R-trees,R-tree,$f_{q,q}$,S_c,S_{fr},F_q)
12. return S_c

Figure 4.3. MTRNN algorithm.

pruning techniques, called closed region pruning and open region pruning, to eliminate all R-tree nodes and points that cannot possibly be MTRNN points. We also prove that both closed and open region pruning techniques do not introduce any false misses in Lemma 1 and Lemma 2 respectively. The refinement step in section 4.3.3 removes all the false hit points by three refinement approaches among which the final approach is to search the multi-type nearest neighbor (MTNN) of each candidate point. If the query point is not one of the points in the MTNN of a candidate point, the candidate point is a false hit and can be eliminated. Otherwise, the candidate point is an MTRNN of the given query point. We prove that the refinement step does not cause any false miss along with the description of the algorithm.

Figure 4.3 presents the overall flow of the MTRNN algorithm and its preparation, filtering and refinement steps. We will discuss these three steps in detail in the following sections. Because feature route is a crucial component in both filtering and refinement, in the following we first discuss the algorithms to find feature routes in the preparation step.

4.3.1 Preparation Step : Finding Feature Routes

A feature route plays an important role in the MTRNN algorithm. We first define feature route and related concepts and then describe our approach of finding the feature routes.

Definition 4 Multi-type route (MTR). *Given k different feature types, a multi-type route is a route that goes through one instance of every feature type.*

Assume there are four feature types F_1, F_2, F_3 and F_4. An MTR could be $R(f_{1,1}, f_{2,1}, f_{3,1}, f_{4,1})$.

Definition 5 Feature route. *Given k different feature types, a feature route is a multi-type route such that the distance from the fixed starting point through all other points in the MTR route is shortest.*

From the MTR $R(f_{1,1}, f_{2,1}, f_{3,1}, f_{4,1})$ illustrated above, we can get four feature routes. Fixing point $f_{1,1}$ and finding the shortest distance from point $f_{1,1}$ through the three other points we get one route. This route, assuming $R(f_{1,1}, f_{4,1}, f_{3,1}, f_{2,1})$ starting from point $f_{1,1}$ with the shortest distance, is a feature route. Starting from each of other three points respectively and finding the route with shortest distance yields four feature routes.

Definition 6 I-distance. *Given a feature route, the (shortest) distance of this route is called an I-distance.*

Definition 7 Feature route point set. *Given a feature route, a feature route point set consists of all points in the feature route.*

A feature route can be identified by a given feature route point set and a fixed starting point. Given a feature route point set containing k points from k different feature types there are k feature routes and k corresponding I-distances starting from each point of the given feature route point set.

The position of a feature route on the search space will affect its filtering ability. Therefore, it is preferred that different feature routes be found for different subspaces of the entire search space. In our algorithms we divide the space into several subspaces

by straight lines intersected at the query point with the same angles between two neighbor lines and find feature routes for each of these subspaces respectively.

Next, we show how to find initial feature route. Figure 4.4 displays the pseudo-code for the greedy MTR finding procedure. This greedy algorithm uses a heuristic method by assuming a fixed order of feature types represented as $< F_q, F_1, \ldots, F_{q-1}, F_{q+1}, \ldots, F_k >$ among which F_i is feature type and greedily find the MTR. A greedy approach is necessary because finding a feature route using the MTNN algorithm is very time-consuming. A greedy approach is also sufficient because the route it finds is used only for pruning purposes. We find the greedy MTR by greedily finding a route for a specified ordered list of features starting from the given query point $f_{q,q}$. A greedy MTR route in terms of k feature types for the given query point $f_{q,q}$ of feature type F_q is found by finding in feature type F_q the nearest neighbor $f_{q,1'}$ of $f_{q,q}$ inside a subspace, and then in feature type F_1, finding the nearest neighbor $f_{1,1'}$ of the point $f_{q,1'}$. This procedure continues until all feature types have been visited. During this procedure, the R-tree index of each feature type is used in the nearest neighbor search algorithm based on work in [49]. All points on the greedy MTR route form a feature route point set. When using a greedy approach to find an MTR we should avoid generating the same MTR more than one time. This is done by making sure that the nearest neighbor of the starting point $f_{q,q}$ is not in the existing feature route set.

Generating one greedy MTR route, thus one feature route point set, for each of a few subspaces may not create large enough pruning regions. For an MTRNN query to generate pruning regions larger enough to filter as many as R-tree nodes and points as possible, it is required to generate enough feature routes. However, there is a tradeoff. The greater the number of feature routes the more expensive the filtering cost due to the greater number of pruning regions generated for each feature route. In our experiments, we show how many feature routes are enough for our filtering algorithm.

There are two approaches to generate the feature routes. The first is to generate m different feature route point sets by finding m NNs of the query point $f_{q,q}$ in feature type F_q and from each of these m NNs greedily find the MTRs to get m greedy MTR routes. This would likely enlarge the pruning regions, and thus increase the filtering ability and reduce the refining cost. The other approach to generate more feature routes, and thus larger pruning regions, is to partition the space into more subspaces.

Algorithm Greedy(R-trees, $f_{q,q}$, S_{fr})
Input : R-trees for each feature, query point $f_{q,q}$, existing feature route set S_{fr}
Output : A feature route point set S_{frps}

```
1.  q = f_{q,q}, S_frps = ∅
2.  For next feature in feature list with predefined order
3.      Remove head from the feature list
4.      Find NN of q in current feature
5.      //Avoid finding the same S_frps twice
6.      If q is f_{q,q} and NN is in S_fr
7.          return ∅
8.      put the NN into S_frps
9.      q = NN
10. return S_frps
```

Figure 4.4. Find greedy MTR.

Because one greedy MTR is found for one subspace, more subspaces means that more greedy MTR are generated. So, more feature routes are then generated. Both approaches have similar effect to increase the filtering ability and reduce cost. In our experiments, we apply the second approach to generate more subspaces and thus feature routes.

Figure 4.5 describes the algorithm for finding feature routes and corresponding I-distances. Since an I-distance is shortest, the route that has length I-distance is a Hamilton Path. We use the existing Hamilton Path-finding algorithm to find the I-distance. As it is known, Hamilton Path-finding problem is NP-complete. For large number of feature types, it is very time-consuming. However, it is acceptable to use the existing algorithm to find exact shortest path for the Hamilton Path-finding in our case since our MTRNN algorithm is only used when the number of feature type is small due to its complexity. As will be discussed later, the shorter the I-distance, the better it is for filtering efficiency.

In the feature route finding algorithm described in Figure 4.5, only the first point of a greedy MTR route is required to be inside a specified subspace because the feature routes generated this way could possibly be shorter. If we require all points in a feature route point set to be inside the same subspace, the feature routes may be longer, which will generate pruning regions with smaller size, thus decreasing

Algorithm FindFeatureRoutes(R-trees, $f_{q,q}$, S_{fr}, S_{sub})
Input : R-trees for each feature, query point $f_{q,q}$, existing feature route set S_{fr}, subspace set S_{sub}
Output : new feature route set S'_{fr}

```
1.  S'_fr = ∅
2.  For each subspace in S_sub
3.      S_frps = Greedy(R-trees, f_q,q, S_fr)
4.      For each point in S_frps
5.          Fix this point as starting point;
6.          Find I-distance and corresponding feature route;
7.          Put the feature route into feature route set S'_fr
8.  return S'_fr
```

Figure 4.5. Find feature routes.

the filtering ability. Although a feature route generated with this strategy may fall into different subspaces, this should not change its filtering ability much statistically, considering that part of any feature route could fall outside the specified subspace and all feature routes are used to generate pruning regions in an R-tree node filtering. It is worth noting that for every greedy MTR route, one feature route point set and k different feature routes are created.

To summarize the concepts of space partitioning and feature routes, Figure 4.6 illustrates a specific space partitioning of six subspaces and some feature routes. Three lines l_1, l_2 and l_3 intersect at the query point $f_{q,q}$, and partition the space around it into six subspaces $S_1, S_2, \ldots, S_6$. For convenience, one of the lines is parallel to the x axis, and the angle between the lines is $360°/6 = 60°$. Please note that the space can be divided into any number of subspaces. Although this specific partitioning scheme with six subspaces in Figure 4.6 is the same as in [58], the MTRNN algorithm does not used the property used in [58]. Starting from each subspace, a greedy MTR route is found. In the figure, sample feature routes of three feature types are given. In this example, not all points on one of the feature routes are inside the same subspace, because the feature route finding algorithm does not guarantee that all points will be inside the same subspace.

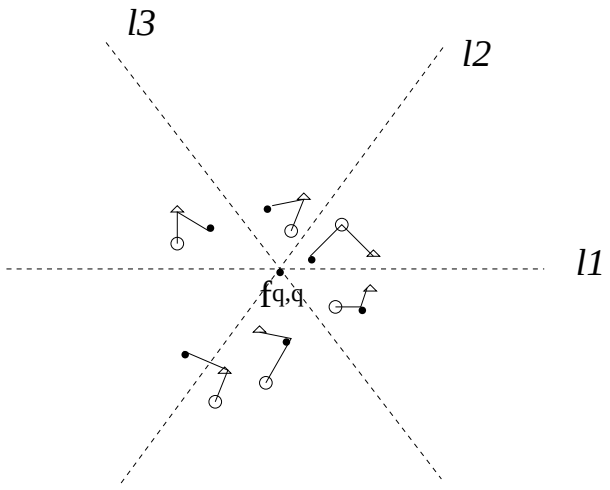


Figure 4.6. Feature routes on the divided space.

4.3.2 The Filtering Step : R-tree Node Level Pruning

After finding feature routes we use two filtering approaches, closed region pruning and open region pruning to prune the search space. In both approaches, feature routes are used to generate pruning regions such that any point inside these regions cannot be MTRNNs, and thus can be filtered without causing any false miss. We begin with the discussion of closed region pruning.

Closed Region Pruning

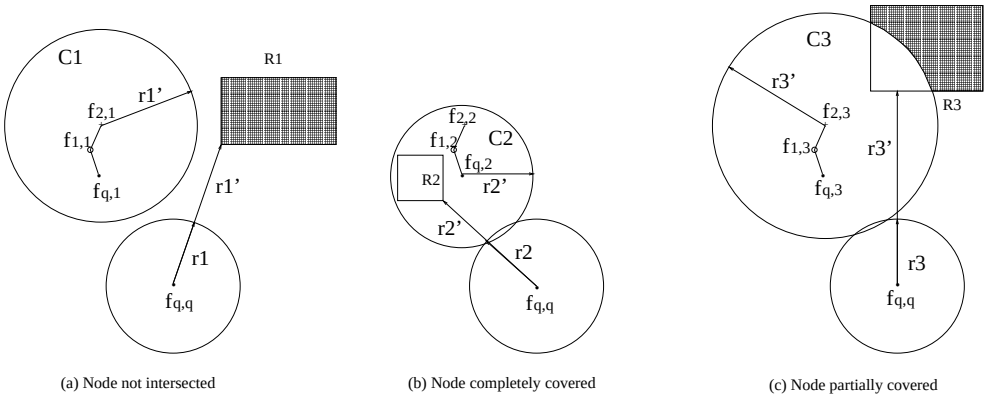


Figure 4.7. Three pruning scenarios.

Definition 8 Closed pruning region. *A closed pruning region is a circle centered at the starting point $f_{i,j}$ in the feature route with radius = MINDIST from the query*

point $f_{q,q}$ to an R-tree node of the queried data − I-distance of the feature route starting at the point $f_{i,j}$.

Closed region pruning generates closed regions for each feature route and one R-tree node and uses these regions to prune the R-tree node. Figure 4.7 illustrates three pruning scenarios. In the figure, $f_{q,q}$ is the query point and R_1, R_2 and R_3 are R-tree nodes that could be either leaf nodes or internal nodes in the R-tree index of the queried data. There are three points inside a feature route point set from three different feature types F_q, F_1 and F_2.

In Figure 4.7, r_i is the length of a feature route and r'_i is the length between the MINDIST from a query point to an R-tree node and r_i. In Figure 4.7a the length of feature route $R(f_{2,1}, f_{1,1}, f_{q,1})$ is r_1 which is the I-distance starting from point $f_{2,1}$ through point $f_{1,1}$ and $f_{q,1}$. During the pruning procedure as shown in Figure 4.7a, first the minimum distance MINDIST from the query point $f_{q,q}$ to the R-tree node R_1 has been calculated. Next the I-distance r_1 of the feature route $R(f_{2,1}, f_{1,1}, f_{q,1})$ was retrieved from the pre-calculated results in preparation step. The MINDIST from $f_{q,q}$ to R_1 is $r_1 + r'_1$ because $r'_1 = MINDIST$ from $f_{q,q}$ to R_1 - r_1. In the following, a circle C_1 is drawn, centered at the starting point $f_{2,1}$ of the feature route $R(f_{2,1}, f_{1,1}, f_{q,1})$. It can be seen that circle C_1 does not intersect the R-tree node R_1. Therefore, the length of the route from a point p_1 inside R_1 to the query point $f_{q,q}$ could be shorter than the distance from the same point through route $R(f_{2,1}, f_{1,1}, f_{q,1})$, i.e., $d(R(p_1, f_{q,q}))$ could be shorter than $d(R(p_1, f_{2,1}, f_{1,1}, f_{q,1}))$, which means that from a point inside R_1 it is possible to find an MTNN that contains the query point $f_{q,q}$. Thus, all points inside R_1 should be evaluated to find whether their MTNNs contain the query point $f_{q,q}$ or not.

In Figure 4.7b, r_2 is the I-distance of route $R(f_{q,2}, f_{1,2}, f_{2,2})$. $r'_2 = MINDIST$ from $f_{q,q}$ to R_2 - r_2 so the MINDIST from $f_{q,q}$ to R_2 is $r_2 + r'_2$. Similarly a circle C_2 centered as $f_{q,2}$ is drawn. At this time it covers the R-tree node R_2 completely. Therefore the length of the route from a point p_2 inside R_2 through $R(f_{q,2}, f_{1,2}, f_{2,2})$ could not be longer than the distance from the same point to the query point $f_{q,q}$, i.e., $d(R(p_2, f_{q,2}, f_{1,2}, f_{2,2})) < d(R(p_2, f_{q,q}))$, which means from any point inside R_2 we could not find an MTNN that contains the query point $f_{q,q}$. Thus, the entire node R_2 can be pruned.

In Figure 4.7c, r_3 is the I-distance of route $R(f_{2,3}, f_{1,3}, f_{q,3})$. $r'_3 = MINDIST$ from $f_{q,q}$ to R_3 - r_3 so the MINDIST from $f_{q,q}$ to R_3 is $r_3 + r'_3$. Another circle C_3 centered at $f_{2,3}$ is drawn and intersects the R-tree node R_3. Therefore the length of the route from a point p_3 inside the intersection of R_3 and circle C_3 through route $R(f_{2,3}, f_{1,3}, f_{q,3})$ could not be longer than the distance from the same point to the query point $f_{q,q}$, i.e., $d(R(p_3, f_{2,3}, f_{1,3}, f_{q,3})) < d(R(p_3, f_{q,q}))$. However, the route from a point p'_3 of R_3 outside the intersection through route $R(f_{2,3}, f_{1,3}, f_{q,3})$ could be longer than $d(R(p'_3, f_{q,q}))$. That means from any point inside the intersection we could not find an MTNN that contains the query point $f_{q,q}$ but from a point outside the intersection an MTNN could be found containing the query point. Thus, the part of page R_3 inside the circle C_3 could be pruned.

From the pruning procedure illustrated in Figure 4.7, we know that the closed pruning region cannot contain any MTRNN. Thus, pruning the closed region won't cause any false miss. The following Lemma 1 formally proves this point.

Lemma 1 *For any point contained inside an R-tree node of the queried data set, if it is also contained inside a closed pruning region it cannot be an MTRNN, and thus can be pruned without causing any false miss.*

Proof Assume p_i is contained in an R-tree node and inside a closed pruning region. Because p_i is contained in the R-tree node, the distance from p_i to the query point is longer than or equals the MINDIST from the query point to the R-tree node. Since the radius of the closed region circle equals the MINDIST from the query point to the R-tree node − the I-distance of the feature route, the distance from p_i that is inside the circle representing the closed pruning region to the starting point of the feature route + I-distance is shorter than MINDIST from the query point to the R-tree node. Therefore, the distance from p_i through the feature route is shorter than the distance from p_i to the query point. As we know that the distance of the MTNN route starting at p_i is shortest among all routes starting from p_i through one instance of each of k feature types, any route from p_i through the query point cannot be the MTNN route, which means the MTNN route from p_i does not contain the query point. So p_i is not an MTRNN of the query point.

Please note that the set of points mentioned in Lemma 1 does not contain any point on the circle of the closed region circle.

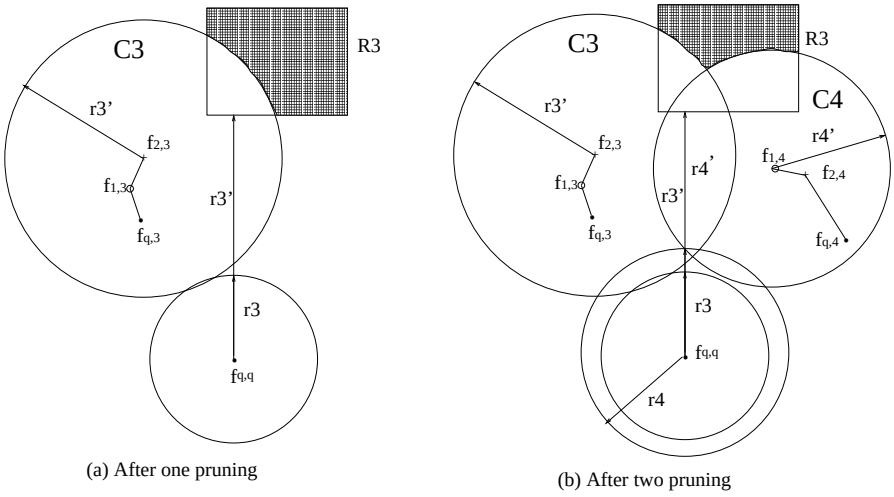


Figure 4.8. Pruning one node.

Figure 4.8 illustrates the pruning process on one R-tree node with two feature routes. Figure 4.8a that is Figure 4.7c shows the pruning result with feature route $R(f_{2,3}, f_{1,3}, f_{q,3})$. The intersected part of R_3 and circle C_3 has been pruned. However, the shaded part may still contain potential MTRNNs as discussed for scenario three in Figure 4.7c. In Figure 4.8b, another feature route $R(f_{1,4}, f_{2,4}, f_{q,4})$ joins in the pruning process. Similar to scenario three in Figure 4.7c, the length of the circle C_4 centered at $f_{1,4}$ is the difference r'_4 between the MINDIST from the query point $f_{q,q}$ to R_3 and the I-distance of feature route $R(f_{1,4}, f_{2,4}, f_{q,4})$. The closed pruning region represented by circle C_4 also prunes part of R-tree node R_3 as shown in Figure 4.8b. For better understanding, we draw two circles centered at $f_{q,q}$ with radius r_3 and r_4 among which r_3 is the I-distance of feature route $R(f_{2,3}, f_{1,3}, f_{q,3})$ and r_4 is the I-distance of feature route $R(f_{1,4}, f_{2,4}, f_{q,4})$. The shaded part of R_3 in Figure 4.8b contains potential MTRNNs. The other part of R_3 cannot contain any MTRNN so it can be pruned without causing any false miss.

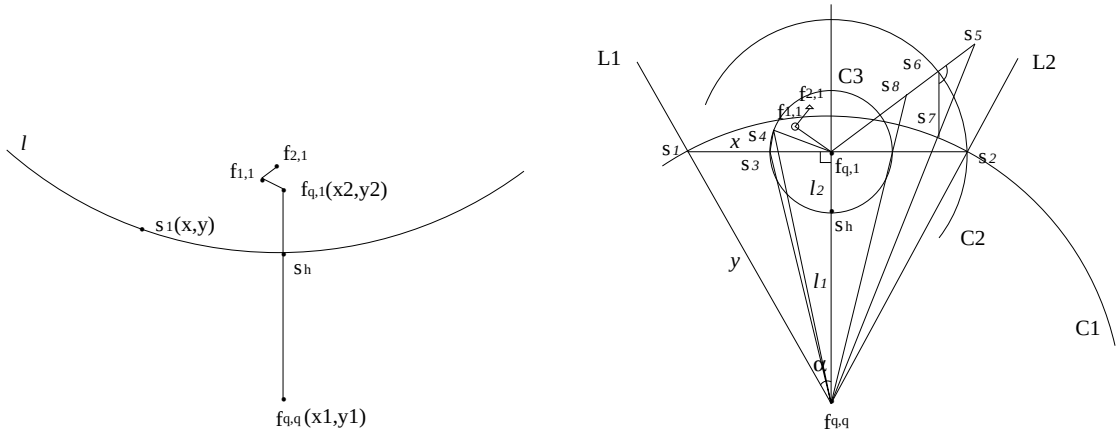
The filtering ability of a feature route in the closed region pruning

As we noted earlier, the filtering ability of a feature route depends on its position relative to the position of the R-tree node to be pruned. The radius of the circle centered at a point on a feature route equals the MINDIST from the query point to an R-tree node - the I-distance of the feature route. If the MINDIST from the query point to the R-tree node is less than the I-distance of a feature route, this feature route will not be used to generate a closed pruning region for this R-tree node and it

will not prune any data point inside this R-tree node. The more the circle covers an R-tree, the better the filtering ability. So, in order to increase the pruning ability of a feature route, it is necessary to calculate its I-distance, which is a Hamilton Path problem. Because the number of feature types is not big and the MTNN finding algorithm is complicated, it is worth calculating the I-distance and having it serve as the length of the feature route. After calculating the radius of the circle, both the circle's size and location are determined on the space. The closer the R-tree node is to the center of a circle, the higher the probability that the circle covers more of the R-tree, and thus the better the filtering ability. Therefore, we can incrementally add more feature routes that are close to an R-tree node into the feature route set and use them to filter the remaining part of the R-tree node and other R-tree nodes later.

Open Region Pruning

Our second pruning approach prunes an open region solely based on each feature route, thus pruning all points and R-tree nodes inside this region. This pruning approach is especially effective when pruning an R-tree node far away from the query point.



(a) Ideal pruning (b) Realistic Pruning

Figure 4.9. Open region pruning.

There are multiple ways to generate an open pruning region. A theoretic maximum open region that is generated from a feature route and can be pruned is represented as a half plane separated by a curve as illustrated in Figure 4.9a. In the figure, $f_{q,q}(x_1, y_1)$ is the query point from feature F_q, $f_{q,1}(x_2, y_2)$ is any point from feature

F_q, $f_{1,1}$ is a point from feature F_1 and $f_{2,1}$ is a point from feature F_2. The space has been divided by curve l into two planes. Plane(l,$f_{q,1}$) represents the plane separated by curve l and containing point $f_{q,1}$ and Plane(l,$f_{q,q}$) represents the plane separated by curve l and containing the query point $f_{q,q}$.

Next, we will define the Plane(l,$f_{q,1}$) such that the distance from any point p inside this plane to $f_{q,1}$ plus the $I-distance$ of feature route $R(f_{q,1}, f_{1,1}, f_{2,1})$ is shorter than the distance from point p to the query point $f_{q,q}$. Therefore, it is impossible that the query point $f_{q,q}$ is on the MTNN of the point p, which means that p is not MTRNN of the query point $f_{q,q}$. Because point p is any point inside Plane(l,$f_{q,1}$), the whole Plane(l,$f_{q,1}$) can be pruned without incurring any false miss.

In the following, we describe how to find curve l so that it can divide the space into Plane(l,$f_{q,1}$) and Plane(l,$f_{q,q}$). In Figure 4.9a curve l and straight line $f_{q,1}f_{q,q}$ intersect at point s_h. $R(f_{q,1}, f_{1,1}, f_{2,1})$ is a feature route and the distance from point s_h to starting point $f_{q,1}$ of the feature route $R(f_{q,1}, f_{1,1}, f_{2,1})$ plus the $I-distance$ of this feature route equals the distance from point s_h to the query point $f_{q,q}$, i.e., $d(R(s_h, f_{q,1}, f_{1,1}, f_{2,1})) = d(R(s_h, f_{q,q}))$. In other words, point s_h divides the line segment $f_{q,1}f_{q,q}$ into two parts so that $d(R(s_h, f_{q,q}))$ - $d(R(s_h, f_{q,1})) = d(R(f_{q,1},f_{1,1},f_{2,1}))$. In the following discussion, we will use h to represent the I-distance of the feature route $R(f_{q,1}, f_{1,1}, f_{2,1})$ in Figure 4.9a.

In order to guarantee that a point inside Plane(l,$f_{q,1}$) is not an MTRNN of the query point $f_{q,q}$, any point $p(x_p, y_p)$ inside Plane(l,$f_{q,1}$) should satisfy the equation $\sqrt{(x_p - x_1)^2 + (y_p - y_1)^2} \geq \sqrt{(x_p - x_2)^2 + (y_p - y_2)^2} + d(R(f_{q,1},f_{1,1},f_{2,1}))$. In this equation $d(R(f_{q,1},f_{1,1},f_{2,1})) = h$. Thus, a point $s_1(x, y)$ on curve l should satisfy the equation $\sqrt{(x - x_1)^2 + (y - y_1)^2} = \sqrt{(x - x_2)^2 + (y - y_2)^2} + h$, which can actually be transformed into a quartic (4-th degree) equation. Because positions of point $f_{q,q}(x_1, y_1)$ and $f_{q,1}(x_2, y_2)$ and $I-distance$ h are known, the curve l is known and divides the space into two planes.

Although curve l in Figure 4.9a can be used to maximally prune R-tree nodes and points, it is not easy to check on which side of the curve l an R-tree node or a point falls. From a practical point of view, a simple representation of an open pruning region should be used in order to prune points and R-tree nodes efficiently. To simplify the point check process, we propose a simpler open region pruning approach based on a simpler region description. In Figure 4.9b, the feature route is $R(f_{q,1}, f_{1,1}, f_{2,1})$, starting at point $f_{q,1}$ with I-distance h. The point s_h divides the line segment $f_{q,q}f_{q,1}$

into two parts with length l_1 and l_2 such that $l_1 - l_2 = h$. Since the positions of points $f_{q,q}$ and $f_{q,1}$ and I-distance h are known, the position of s_h is known, thus the lengths of l_1 and l_2 being also known.

We construct the simple pruning region as follows. In Figure 4.9b we draw a straight line s_1s_2 passing through point $f_{q,1}$ and perpendicular to line $f_{q,q}f_{q,1}$. We then draw line $f_{q,q}s_1$ as line L_1 and line $f_{q,q}s_2$ as line L_2. We take $y - x = h$ in which y is the length of line $f_{q,q}s_1$ and x is the length of line $f_{q,1}s_1$. Since $(l_1+l_2)^2+x^2 = y^2$, we can calculate $x = \frac{2l_1l_2}{l_1-l_2}$ and $y = \frac{{l_1}^2+{l_2}^2}{l_1-l_2}$. Because l_1 and l_2 are known, x and y are known. Therefore, the positions s_1 and s_2 are also known.

In formulas $x = \frac{2l_1l_2}{l_1-l_2}$ and $y = \frac{{l_1}^2+{l_2}^2}{l_1-l_2}$, l_2 is between 0 and $\frac{len(f_{q,1}f_{q,q})}{2}$. When l_2 is 0, which means the I-distance of the feature route $R(f_{q,1}, f_{1,1}, f_{2,1})$ is equal to or longer than len($f_{q,1}f_{q,q}$), no point can be pruned by using the open region generated based on this feature route. When l_2 is $\frac{len(f_{q,1}f_{q,q})}{2}$, which means the I-distance h of this feature route is 0, the MTRNN problem reduces to the classic RNN problem for this feature route. Then the perpendicular bisector $\perp(f_{q,1}, f_{q,q})$ divides the data space into two half planes: one that contains $f_{q,1}$ (Plane($\perp(f_{q,1}, f_{q,q})$,$f_{q,1}$)), and one that contains $f_{q,q}$ (Plane($\perp(f_{q,1}, f_{q,q})$,$f_{q,q}$)). No point in Plane($\perp(f_{q,1}, f_{q,q})$,$f_{q,1}$) can be an RNN or an MTRNN of $f_{q,q}$ and thus all R-tree nodes and points in Plane($\perp(f_{q,1}, f_{q,q})$,$f_{q,1}$) can be pruned.

Next we prove that all points in the open pruning region formed by lines L_1 and L_2 excluding the triangle $f_{q,q}s_1s_2$ in Figure 4.9b can be pruned. That is, the distance from any point inside this region to the query point $f_{q,q}$ is longer than the distance of the point to point $f_{q,1}$ plus the I-distance h of the feature route $R(f_{q,1}, f_{1,1}, f_{2,1})$ starting at point $f_{q,1}$.

Lemma 2 *No point inside an open pruning region defined by a feature route and the query point can be an MTRNN, and thus can be pruned.*

Proof The open pruning region containing point $f_{q,1}$ is formed by lines L_1 and L_2, excluding triangle $f_{q,q}s_1s_2$. It is divided into three parts as shown in Figure 4.9b. The first part (part 1) is the open pruning region outside circle C_2. The second part (part 2) is the intersection of circle C_2 and open pruning region, excluding the circle C_1. The third part (part 3) is the intersection of circle C_1 and the open pruning region. We prove the lemma by demonstrating that any point in any part of the open pruning region can be pruned without causing any false miss.

Algorithm OneNodePrune(R-trees,R-tree,S_{fr},$f_{q,q}$)
Input : R-trees for all feature types, R-tree node to be pruned, *feature route set* S_{fr}, query point $f_{q,q}$
Output : Empty set, R-tree node or candidate MTRNNs

```
1.  Calculate mindist = the MINDIST from the query point f_{q,q} to R-tree node;
2.  S'_{fr} = S_{fr}, NoCenterCreated = true;
3.  For each feature route in S'_{fr}
4.      Calculate difference r between mindist and I-distance of feature route;
5.      Form a closed pruning region;
6.      Form an open pruning region
7.  If R-tree node is entirely contained inside pruning regions
8.  Then return empty set
9.  Else If R-tree node is an internal node
10.     Then return R-tree node
11.     Else If NoCenterCreated
12.         //Prune same R-tree node with newly generated feature route sets
13.         Then NoCenterCreated = false;
14.             Find center s_c of the R-tree node and put subspace containing s_c
15.                 into S_{sub};
16.             S'_{fr} = FindFeatureRoutes(Rtrees,s_c,S_{fr},S_{sub});
17.             S_{fr} = S'_{fr} ∪ S_{fr};
18.             Goto 3
19.         Else return All points inside the R-tree node but outside all the
20.             pruning regions
```

Figure 4.10. One node pruning algorithm.

In Figure 4.9b, s_1 and s_2 are positioned on the circle C_1 centered at $f_{q,q}$ with radius y or $len(f_{q,q}s_1)$ and also on the circle C_2 centered at $f_{q,1}$ with radius x or $len(f_{q,1}s_1)$. The radius of the smallest circle C_3 centered at $f_{q,1}$ is len$(f_{q,1}, s_4)$ in which s_4 is any point inside part 3.

First, assume a point s_5 in part 1 is outside of the circle C_2. We need to prove $d(R(s_5, f_{q,1}, f_{1,1}, f_{2,1})) < d(R(s_5, f_{q,q})$. It can be seen that $d(R(s_5, f_{q,1}, f_{1,1}, f_{2,1})) = d(R(s_5, s_6)) + d(R(s_6, f_{q,1})) + d(R(f_{q,1}, f_{1,1}, f_{2,1}))$ and $d(R(s_5, f_{q,q}) = d(R(s_5, s_7) + d(R(s_7, f_{q,q})$. Because len$(f_{q,q}s_7)$ = len$(f_{q,q}s_2)$, len$(f_{q,1}s_6)$ = len$(f_{q,1}s_2)$ and len$(f_{q,q}s_2)$ - len$(f_{q,1}s_6)$ = h, len$(f_{q,q}s_7)$ - len$(f_{q,1}s_6)$ = $d(R(f_{q,1}, f_{1,1}, f_{2,1}))$. Therefore, we only need to prove len$(s_5s_7) \geq$ len(s_5s_6). It is easy to prove angle $\angle s_7s_6s_5 \geq 90°$ so len(s_5s_7) $\geq$ len(s_5s_6). So, s_5 can be pruned.

When a point s_8 in part 2 is inside the circle C_2 but outside the circle C_1, $f_{q,q}s_8$ is longer than $f_{q,1}s_2$ but $f_{q,1}s_8$ is shorter than $f_{q,1}s_2$. Therefore d(R($s_8, f_{q,1}f_{1,1}, f_{2,1}$)) < d(R($s_8, f_{q,q}$)) and the point s_8 can be pruned.

Finally, assume a point s_4 in part 3 is inside the circle C_1. Since $\angle s_3s_4f_{q,1} > \angle s_3s_4f_{q,q}$ and $\angle s_4s_3f_{q,1} = \angle s_3s_4f_{q,1}$ we have $\angle s_4s_3f_{q,1} > \angle s_3s_4f_{q,q}$. Thus, $\angle s_4s_3f_{q,q} > \angle s_3s_4f_{q,q}$. Therefore we derive len($s_4f_{q,q}$) > len($s_3f_{q,q}$). Since we know len($s_3f_{q,q}$) - len($s_3f_{q,1}$) > h, then len($s_4f_{q,q}$) - len($s_4f_{q,1}$) > h. So, the distance from point $s4$ to $f_{q,1}$ plus I-distance starting at $f_{q,1}$ is shorter than the length of $s_4f_{q,q}$, i.e., d(R($s_4, f_{q,1}f_{1,1}, f_{2,1}$)) < d(R($s_4, f_{q,q}$)). Therefore, point s_4 can be pruned.

Since Lemma 2 proves that no point inside an open region can be an MTRNN, pruning the whole open region won't introduce any false miss.

Figure 4.10 presents the pseudo-code for the algorithm applying both closed region and open region pruning. If an R-tree node is entirely contained inside pruning regions, the R-tree node can be filtered "safely" and the output set is empty, meaning the MTNN from a point in the R-tree node cannot contain the given query point and can safely be filtered without causing a false miss. Otherwise, if the R-tree node is an internal node output the R-tree node itself. The filtering algorithm will then visit all child nodes of this internal R-tree node later. If the R-tree node is a leaf node and not all of it is covered by pruning regions, use the center point of the node as a query point to find in specified subspaces a greedy MTR route whose points form a new feature route point set. Next, find feature routes and use these new feature routes to generate new pruning regions and try to prune the R-tree node. Because a center point of an R-tree node can be anywhere either close to or far away from the query point, we only find new greedy MTR routes whose starting point is inside the subspace containing the center point in order to avoid generating too many feature routes.

Filtering Algorithm

The complete filtering step should search all candidate points in the data set against which the query is issued. Since R-trees are widely used in spatial databases, we assume an R-tree index is available for each feature type and the queried data set. Our filtering algorithm utilizes R-tree indexes to generate closed and open pruning regions that are used to filter as many as R-tree nodes and points in the queried data set. The example in Figure 4.11 illustrates how the filtering algorithm works.

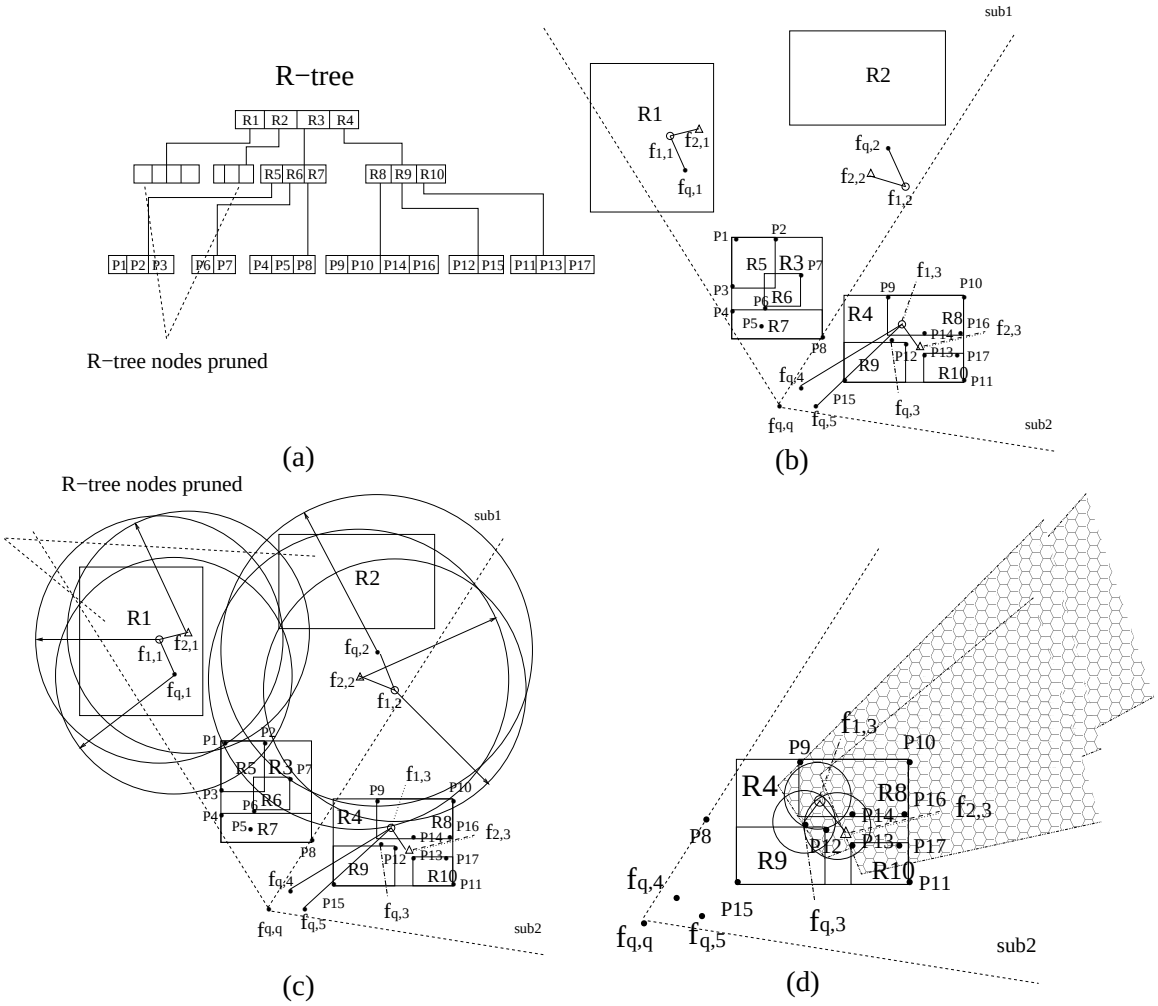


Figure 4.11. A filtering example.

In this example, the R-tree nodes at the first level contain R_1, R_2, R_3 and R_4. Figure 4.11a gives the R-tree index of the queried data set. The query point $f_{q,q}$ is of feature type F_q. For simplicity, we don't draw R-tree nodes of feature data sets and only illustrate the filtering process in two subspaces sub_1 and sub_2.

Initially, as shown in Figure 4.11b we use 2 NN strategy and find in feature type F_q two nearest neighbors $f_{q,1}$ and $f_{q,2}$ of the query point $f_{q,q}$ inside the subspace sub_1. Then we find nearest neighbor $f_{1,1}$ of $f_{q,1}$ and nearest neighbor $f_{1,2}$ of $f_{q,2}$ in feature type F_1. Finally we find nearest neighbor $f_{2,1}$ of $f_{1,1}$ and nearest neighbor $f_{2,2}$ of $f_{1,2}$ in feature type F_2. So far, we get two feature route point sets $\{f_{q,1}, f_{1,1}, f_{2,1}\}$ and $\{f_{q,2}, f_{1,2}, f_{2,2}\}$. For each feature route point set, we calculate the I-distances of the feature routes starting from each point in the feature route point set. For example,

the I-distance of the feature route starting at point $f_{q,1}$ in the feature route point set $\{f_{q,1}, f_{1,1}, f_{2,1}\}$ is $d(R(f_{q,1}, f_{1,1}, f_{2,1}))$. Similarly we find feature route point sets $\{f_{q,4}, f_{1,3}, f_{2,3}\}$ and $\{f_{q,5}, f_{1,3}, f_{2,3}\}$ in subspace sub_2. Because either I-distance of any feature route from sets $\{f_{q,4}, f_{1,3}, f_{2,3}\}$ and $\{f_{q,5}, f_{1,3}, f_{2,3}\}$ is longer than MINDIST from $f_{q,q}$ to an R-tree node (R_3 or R_4) or R-tree nodes (R_1 and R_2) have already been pruned by other pruning regions, sets $\{f_{q,4}, f_{1,3}, f_{2,3}\}$ and $\{f_{q,5}, f_{1,3}, f_{2,3}\}$ are not used to prune any R-tree node. For simplicity we ignore them and do not draw pruning regions generated from them.

The next step illustrated in Figure 4.11c shows the closed region pruning with feature routes found so far. It starts with calculating the MINDIST from the query point $f_{q,q}$ and R-tree node R_1. Following this step, we calculate the difference between the MINDIST from the query point $f_{q,q}$ and R-tree node R_1 and the I-distance of $R(f_{q,1}, f_{1,1}, f_{2,1})$ and draw a circle centered at $f_{q,1}$ with this difference. We repeat these steps for all points in the feature point route set $\{f_{q,1}, f_{1,1}, f_{2,1}\}$ and get three feature routes and circles. The R-tree node R_1 is completely covered by these circles so it cannot contain any MTRNN and can be pruned. Similarly R-tree node R_2 can be pruned completely. However, R-tree node R_3 is only partially covered by the circles. Since from the center point of R_3 no new feature route set is found in the subspace sub_1, we don't try to prune R-tree node R_3 again. Thus, it traverses down node R_3 to visit R_5, R_6 and R_7. At R_5, points p_1 and p_2 are inside the circles and can be pruned but point p_3 is left as a potential MTRNN. Similarly point p_6 in node R_6 and points p_4, p_5 and p_8 in node R_7 are potential MTRNNs. Since R-tree nodes R_1 and R_2 were pruned earlier and R-tree node R_3 is not pruned by the open pruning regions, the open pruning regions generated from feature point set $\{f_{q,1}, f_{1,1}, f_{2,1}\}$ and $\{f_{q,2}, f_{1,2}, f_{2,2}\}$ have not been drawn.

In this example, R-tree node R_4 is not entirely pruned by all existing closed and open pruning regions (not shown in this figure for simplicity) and only points p_9 and p_{10} in node R_8 were pruned. Figure 4.11d gives the example about how to prune points inside R-tree node R_4 of region sub_2 by using closed and open region pruning techniques generated from a new query point. For simplicity, only subspace sub_2 is shown in Figure 4.11d. We take the center of node R_4 as the new query point and find a greedy route $R(f_{q,3}, f_{1,3}, f_{2,3})$ from it. As before, three circles are drawn. Point p_{14} in node R_8, point p_{12} in node R_9 and point p_{13} in node R_{10} are then pruned by the new circles. So far, the only potential MTRNNs are point p_{15} in node R_9,

point p_{16} in R_8 and points p_{11} and p_{17} in node R_{10}. Now we apply the open region pruning approach. The open pruning region is the region filled with hexagons. At this time, points p_{16} in R_8 and p_{17} in node R_{10} fall into the open pruning regions so they are pruned. Finally only point p_{15} in node R_9 and point p_{11} in node R_{10} are left as candidate MTRNNs.

```
Algorithm Filtering(R-trees,R-tree,S_fr,f_q,q,p_c)
Input : R-trees for all feature types, R-tree for data set being queried,
        Feature Routes S_fr, Query Point f_q,q, center point p_c
Output : Empty set, R-tree node or candidate MTRNN set S_c
 1.  R-tree = OneNodePrune(R-trees,R-tree,S_fr,f_q,q)
 2.  If R-tree is empty set
 3.  Then return empty set
 4.  Else If R-tree is an internal node
 5.      Then If p_c is not NIL
 6.          Then return R-tree
 7.          Else Find center p_c of R-tree node and put subspace containing p_c
 8.                  into S_sub
 9.               S'_fr = FindFeatureRoute(R-trees,p_c,S_fr,S_sub);
10.               S_fr = S'_fr ∪ S_fr;
11.               //Prune R-tree node with new feature route set
12.               R-tree = Filtering(R-trees,R-tree,S'_fr,f_q,q,p_c)
13.               If R-tree is empty set
14.               Then Return empty set
15.               //Prune child nodes of the R-tree node
16.               For each child node of R-tree
17.                   Add Filtering(R-trees,child node,S_fr,f_q,q) into PointSet
18.     Else Add R-tree into PointSet
19. return PointSet
```

Figure 4.12. Filtering algorithm.

Figure 4.12 gives the pseudo-code of the filtering algorithm. For an internal R-tree node, the Filtering function is called twice. At the first call, center point p_c is empty and the existing feature route set is used to prune the R-tree node. At the second call, center point p_c is found and the newly generated feature route set is used to prune the R-tree node. If the R-tree node still cannot be pruned completely, each child node is pruned with all feature routes including the newly generated ones. After

filtering, most of the points in the queried data set are safely pruned without causing any false miss and a candidate point set S_c containing all potential MTRNNs has been generated.

4.3.3 Refinement Step: Removing False Hit Points

The refinement step further eliminates points in the MTRNN candidate set S_c so that only qualified MTRNNs will remain. Three refinement approaches are applied to guarantee all false hits will be eliminated.

Figure 4.13 shows the pseudo-code for the complete refinement step.

The first approach uses existing feature routes to eliminate false hits from candidate MTRNNs. After filtering, we have a set S_c of candidate MTRNN points and a set S_{fr} of feature routes. Since the I-distances of feature routes have already been calculated, they can be directly used to eliminate false hits that cannot be real MTRNNs. If the minimum distance $mindist$, distance from an MTRNN candidate point p to the starting point of one feature route plus the I-distance of the feature route, is shorter than the distance d_1 from p to the query point $f_{q,q}$, which means the MTNN distance from the point p is shorter than the distance d_1 and the MTNN of point p cannot contain the query point $f_{q,q}$, the point p cannot be an MTRNN of the query point $f_{q,q}$ and can be pruned. In other words, this approach does not introduce any false miss.

In the second approach, a greedy MTR route and corresponding feature route point set are calculated if an MTRNN candidate point p cannot be pruned by using the first approach. Note that query point $f_{q,q}$ is considered as a point in the data set of feature type F_q when finding this greedy MTR route. From the new feature route point set, a new set of feature routes can be found. If the minimum distance $mindist$, distance from an MTRNN candidate point p to the starting point of one new feature route plus the I-distance of the new feature route, is shorter than d_1, this point could be pruned. Similar to the first approach, the second approach does not cause any false miss. Since it is possible there are some points close to p in the candidate set, this newly found feature route point set could be useful to prune these points (and other points); thus the new point set from this greedy MTR route is added into the set of feature route point set S_{frps} and all I-distances for all feature routes from this feature route point set are saved for future pruning.

If an MTRNN candidate point p cannot be pruned by the first two approaches,

Algorithm Refinement(R-trees,R-tree,$f_{q,q}$,S_c,S_{fr},F_q)
Input : R-trees for each feature type, R-tree for data set being queried, Query Point $f_{q,q}$, a candidate MTRNN set S_c, the *feature route* set S_{fr}, the query feature type F_q
Output : MTRNN set S_c

```
1.  mindist = ∞
2.  For each point p in set S_c
3.      Calculate distance d_1 from point p to the query point f_{q,q};
4.      For each feature route fr in S_fr
5.          Calculate distance d_2 from point p to the starting point of feature
6.              route fr;
7.          if mindist > d_2+ I-distance of fr
8.              mindist = d_2+ I-distance of fr
9.              If mindist < d_1
10.                 Eliminate point p from set S_c;
11.                 goto 1
12.     S_frps = Greedy(R-trees, p, S_fr);
13.     Calculate I-distances for all feature routes S'_fr starting from all points
14.         in S_frps;
15.     For each feature route fr' in S'_fr
16.         Calculate distance d_3 from point p to the starting point of feature
17.             route fr';
18.         if mindist > d_3+ I-distance of fr'
19.             mindist = d_3+ I-distance of fr'
20.             If mindist < d_1
21.                 Eliminate point p from set S_c;
22.                 Put the new feature routes S'_fr into S_fr;
23.                 goto 1
24.     mtnn = MTNN(R-trees, R-tree, mindist, p, f_{q,q}, F_q);
25.     If f_{q,q} is not in mtnn
26.         Eliminate point p from set S_c;
27.         Calculate I-distances for all feature routes starting from all points
28.             in mtnn;
29.         Put the new feature routes into S_fr;
30. return MTRNN set S_c
```

Figure 4.13. Refinement algorithm.

an MTNN algorithm that utilizes R-tree index of each feature type is applied to calculate the real MTNN for this point p. As in the second approach, query point $f_{q,q}$ is considered as a point in the dataset of feature type F_q. After finding MTNN of the point p, we get a set of MTNN points and a corresponding MTNN route. If query point $f_{q,q}$ is in the MTNN of the point p, then point p is an MTRNN of this query point. Otherwise, p is eliminated from S_c. This approach does not cause any false miss. From the MTNN algorithm in [40] we know that the MTNN route from the point p is shortest among all possible routes from p going through each point from each different feature types. If the query point $f_{q,q}$ is on this MTNN route or, in other words, in the point set of MTNN, this point p is an MTRNN of the query point $f_{q,q}$ according to the problem definition formalized in section 4.2. Thus, this third approach does not introduce false miss.

Figure 4.14 shows the pseudo-code of the MTNN algorithm, which is adapted from the algorithm described in [54] and [40]. The initial greedy distance dis is the minimum distance of all routes from point p through all feature routes. The major adaptation occurs during partial route growing to the feature type that the query point $f_{q,q}$ belongs to. If none of the current partial routes for a specific permutation contains the query point $f_{q,q}$, it is safe to stop searching for this permutation. Another enhancement is to mark the partial route ending with query point $f_{q,q}$ after growing the partial route to the feature type. Later, if this marked partial route is not used to grow any further partial routes, the searching for this permutation can be safely stopped. After all features in a permutation are visited, a potential MTNN is generated. After all permutations of all feature types are searched, a real MTNN is generated.

It is worth discussing when the filtering and refinement algorithms fail pruning any point thus work as the naive baseline algorithm. Normally when all queried data points are far away from the query point $f_{q,q}$ and all feature data points that are clustered, our filtering and refinement algorithms may fail or have limited ability to prune points. It means that it is likely the distance from a queried point p to the query point $f_{q,q}$ is longer than the distance of a feature route plus the distance from the start point of the feature route to point p. If this happens, point p cannot be pruned.

Algorithm MTNN(R-trees,R-tree,dis,q,$f_{q,q}$,F_q)
Input : R-trees for each feature type, R-tree index root of queried data set,
potential MTRNN point q, distance dis, Query Point $f_{q,q}$, feature type F_q
of query point
Output : MTNN

```
1.  MTNN = ∅
2.  Prune all R-tree nodes not intersected by the circle centered at q with radius dis
3.  For each permutation of all features
4.  //For simplicity assume permuation is (1, 2, ..., k)
5.      dis_1 = dis
6.      CurFT = k
7.      For each point p in data set of feature type CurFT
8.          If (d(R(p, q)) < dis_1)
9.              Put R(p) into partial route set S
10.     For i = k − 1 to 1
11.         If CurFT is F_q and f_{q,q} is not in S
12.             return empty
13.         CurFT = i
14.         For each point p' in CurFT
15.             Grow each partial route of S by adding p' to the head
16.             if(Length of new partial route +d(R(p', q)) < dis1)
17.                 put new partial route into partial route set S_1;
18.             Put partial route with shortest length in S_1 into partial route set S_2;
19.         S = S_2;
20.         Find route in S with shortest distance dis_2;
21.         dis_1 = dis − dis_2
22.     dis =the distance of current shortest route
23. Find MTNN in route of S with shortest length
24. return MTNN
```

Figure 4.14. Adapted MTNN algorithm.

4.4 Complexity Analysis

In this section we study the complexity of the baseline algorithm and our proposed MTRNN algorithm. Our analysis is based on the cost model for nearest neighbor search in low and medium dimensional spaces devised in the work [63] by Tao *et al.*. In the following we compute the expected time complexity in term of distance calculations required to answer an MTRNN query.

Assume that the points of a queried data set and each feature data set are uniformly distributed in a unit square universe. The number of queried data points is N and each feature data set contains M data points. Similarly to [54], we derive formulas for the following distances

1. The expected distance δ between any pair of points each from a different feature
2. The expected *feature route* distance E_{fr}

Because the cardinality of a feature data set is M and data are uniformly distributed in the unit square universe, it is expected to have $\sqrt{M}$ points along a direction of x or y axis. For two data sets from two different feature types, there are expected $\sqrt{2M}$ points. Because we assume the data are in the unit square universe, the expected distance between any pair of points each from different features is $\delta = \frac{1}{\sqrt{2M}}$.

If there are k points each from a different feature type in a feature route, the expected feature route distance of the feature route is $E_{fr} = (k-1)\delta = \frac{k-1}{\sqrt{2M}}$.

4.4.1 Cost of Baseline Algorithm

Since we use the algorithm R-LORD in [54] to find MTRNN for one permutation of feature types, for example, $< F_1, F_2, \ldots, F_k >$, the cost of the baseline algorithm on one queried data point is just the sum of the R-LORD algorithm cost for all permutations.

Cost of R-LORD algorithm consists of two components, the distance calculation of R-tree node access and distance calculation of point search within the range of each iteration. In the following, we discuss these two components derived by Sharifzadeh *et al.* [54].

As stated in [54] "for each accessed node, R-LORD performs an $O(1)$ $MINDIST$ computation. Therefore, the complexity of each R-tree traversal is the same as the

number of node accesses during the traversal." Thus, the expected number of R-tree nodes, NA, is used to represent the distance calculation of R-tree node access. Following the cost mode proposed in [63], the expected number of node accesses is given as

$$NA = \sum_{i=1}^{h-1} (n_i \times P_{NA_i}) \tag{4.1}$$

In this formula, h is the height of R-tree, P_{NA_i} is the probability of accessing a node at level i, and n_i is the total number of nodes at level i. Given total number of points in a data set, the capacity of R-tree node and the average fan-out of R-tree node, h and n_i can be easily derived [63]. For the P_{NA_i} estimation, it is needed to identify the search region. As derived in [54], the expected ranges for iteration 1 is $k \times \delta$ and for all other following iterations are $(k - i + 2) \times \delta$. Therefore, P_{NA_i} can be easily derived [63].

As LORD algorithm, R-LORD performs the same set of distance calculation for the point search for the chosen point set, so the second part of the R-LORD cost is $C_{lord} - kM$ in [54]. kM is removed from the C_{lord} because the algorithm of finding whether points are within the range in LORD is replaced by R-tree node pruning in R-LORD.

The components that should be considered when deriving cost formula for C_{lord} are the expected number of partial routes, the current search range T_v ($dis1$ in Figure 4.14) and the expected number of points $\pi {T_v}^2 M$ [63] in a feature type that are closer to the starting point than current search range T_v and will be examined in an iteration.

For the initialization step, the current search range T_v is the length of greedy route $T_c = k\delta$ (dis in Figure 4.14), the expected number of partial routes is $\pi k^2 \delta^2 M$ and the expected number of points to be examined is M. For the first iteration, the current search range T_v decreases to $(k-1)\delta$, the expected number of partial routes is updated to $\pi(k-1)\delta^2 M$ and the expected number of points to be examined is $\pi(k-1)^2\delta^2 M$. For each iteration, all the parameters are derived and summarized in Table 4 in work [54]. Finally the cost formula of LORD is derived as follows

$$C_{lord} = O(kM + k^5) \tag{4.2}$$

Therefore, the expected cost of R-LORD can be given as

$$C_{rlord} = \sum_{i=1}^{k} NA(i) + (C_{lord} - kM) = \sum_{i=1}^{k} NA(i) + O(k^5) \tag{4.3}$$

where $NA(i)$ is distance calculation of R-tree node access, that is, the expected number of nodes, accessed in iteration i in R-LORD algorithm [54].

Since the expected length of T_c [54] used in the R-LORD algorithm is the same for all permutations under our assumption and the total number of permutations is $k!$, the cost for one queried point is:

$$C_{1-point} = k! \times (\sum_{i=1}^{k} NA(i) + O(k^5)) \tag{4.4}$$

Therefore, the total cost of the baseline algorithm for all queried points is:

$$C_{baseline} = k! \times O(N) \times (\sum_{i=1}^{k} NA(i) + O(k^5)) \tag{4.5}$$

4.4.2 Cost of MTRNN Algorithm

The efficiency of the MTRNN algorithm is primarily based on the filtering ratio, i.e., the number of candidate MTRNNs after filtering. A good filtering algorithm should dramatically reduce the number of candidate MTRNN points and thus the overall cost of the algorithm. On the other hand, the filtering cost is immaterial when the number of feature types increases, which means the cost of the refinement step is the dominant cost in the MTRNN algorithm. Therefore, we analyze the cost of the refinement step and use it as the total cost of the MTRNN algorithm.

We assume that the feature routes are distributed uniformly in every direction from the query point $f_{q,q}$. In order to find the filtering ratio, we should calculate the area of the closed and open pruning regions and then derive the expected number of candidate MTRNN points that fall outside the pruning regions, based on the assumption of uniform data distribution.

For both of the closed and open region pruning approaches discussed in section 4.3, a feature route starting inside a circle, say C_1, with radius r_1 of length E_{fr}, doesn't have any pruning ability. In other words, no queried data points inside circle C_1 whose radius is r_1 of length E_{fr} can be pruned, so these points are included as the minimum set of points in candidate MTRNN set S_c. Our filtering algorithm cannot

prune any point in this minimum set.

Assume that the number of feature route is l and that these feature routes start outside circle C_1 but inside another circle, say C_2, with radius r_2. The area outside C_1 but inside C_2 is $\pi {r_2}^2 - \pi {r_1}^2$. Since all points from all feature types inside this region are expected to be starting points of feature routes we have $\frac{1}{\pi {r_2}^2 - \pi {r_1}^2} = \frac{kM}{l}$, which gives $r_2 = \sqrt{\frac{2l+\pi k(k-1)^2}{2\pi kM}}$. Assume that the expected distance from the query point $f_{q,q}$ to the starting point of a feature route is r. We have $\pi {r_2}^2 - \pi r^2 = \pi r^2 - \pi {r_1}^2$ so the expected distance $r = \sqrt{\frac{{r_2}^2 + {r_1}^2}{2}} = \sqrt{\frac{l+\pi k(k-1)^2}{2\pi kM}}$.

Next we discuss the area covered by an open pruning region. We first calculate the area of triangle $f_{q,q}S_1S_2$ in Figure 4.9b. In the figure, $f_{q,q}F_{q,1}$ is just the expected distance r from the query point $f_{q,q}$ to the starting point of a feature route, so $h = r - r_1 = \frac{\sqrt{2(l+\pi(k-1)^2)} - \sqrt{\pi(k-1)^2}}{\sqrt{2\pi kM}}$. From Figure 4.9b, we have $y - x = h$ and $r^2 + x^2 = y^2$ so $x = \frac{r^2 - h^2}{2h}$. Since r and h are known values, x is also known.

Therefore, inside a region formed by $f_{q,q}L_1$ and $f_{q,q}L_2$, the area of the triangle region that is not covered by this open pruning region is xr. We assume that there are enough feature routes such that the covered areas of the open pruning regions touch each other. Therefore, the regions not covered by open pruning regions are of area xrl.

For one feature route, a closed region can cover a region with expected area $\pi(r - r_1)^2$. However, half of the region was previously covered by an open pruning region, so one closed region covering a region that was not covered by the pruning regions has area $\frac{\pi(r-r_1)^2}{2}$. Therefore, the total area covered by all the closed pruning regions is $\frac{\pi(r-r_1)^2}{2} l$ for l feature route.

So far, we can calculate that the total area that was not covered by open and closed regions is $A_{NL} = xrl - \frac{\pi(r-r_1)^2}{2} l$. This area is the lower bound of the area that was not covered when we assume the open pruning regions are touching. As more and more feature routes are added, the open pruning regions will overlap each other and closed pruning regions will cover more area outside circle C_1. Finally only a region with area $A_{NU} = \pi {r_1}^2$ is not covered. This is the upper bound of the area that was not covered.

When the open regions are not touching each other, the regions that can be pruned are just the sum of all individual open and closed pruning regions. We ignore the formula for this situation.

After deriving the area of regions A_N that were not covered we can easily derive the number of points N_{sc} in the candidate MTRNN set S_c by applying formula $\frac{1}{A_N} = \frac{N}{N_{sc}}$. So, the $N_{sc} = A_{NU}N$ for the upper bound and $N_{sc} = A_{NL}N$ for the lower bound and the total computation for MTRNN algorithm is $N_{sc}C_{rlord}$. Since the number of feature route does not increase with the number of data points in different features and queried data set, it is considered as constant in the complexity analysis and ignored in the cost model. Our experiment results in sections 4.5.3 and 4.5.3 also confirm that this feature route number is constant. Because all the components for the derivation of the asymptotic upper bound are given above, we ignore details for simplicity. It is easy to derive that the MTRNN algorithm cost for both lower bound covered area and upper bound covered area in terms of asymptotic upper bound is

$$C_{MTRNN} = O(\frac{Nk^2}{M}) \times C_{1-point} = k! \times O(\frac{Nk^2}{M}) \times (\sum_{i=1}^{k} NA(i) + O(k^5)) \qquad (4.6)$$

Although C_{MTRNN} is a factorial function in terms of k, $k!$ is not very big when k is not big, which is the case that MTRNN algorithm applies to.

4.5 Experimental Evaluations

We had two overall goals for our experiments: 1) to evaluate the performance and scalability of the MTRNN algorithm for the MTRNN query and 2) to evaluate the impact of our multi-type approach compared to traditional RNN query methods in terms of number and identity of RNNs returned.

4.5.1 Settings

Experiment Platform Our experiments were performed on a PC with two 2.33 GHz Intel Core 2 Duo CPUs and 3GByte memory running Windows XP SP3 operating system. All algorithms were implemented in Java programming language with Eclipse 3.3.2 as the IDE and JDK 6.0 as the running environment.

Experimental Data Sets We evaluated the performance of the MTRNN algorithm with both synthetic and real datasets.

- Synthetic data sets: All synthetically generated data points were distributed over a 1000X1000 plane. To evaluate the effects of spatial distribution, one

queried data set was generated with random distribution (denoted as RAN) and the second with clustered distribution (denoted as CLU). The data points for different feature types were generated separately, resulting in different distributions in space for each type. The CLU dataset comprised 50 to 100 clusters of data points from all the multiple feature types as well as the queried data.

- Real data sets: We used two real data sets in our experiments, denoted as CA and NC. CA was converted from a California Road Network and POI spatial data set [36]. The queried data and data of all feature types were selected from the Road Network nodes and POIs respectively. For the NC data set, the queried data and features were converted from the North Carolina (NC) Master Address Database Project [1] and a GPS POI data set [2]. Both real datasets contained multiple different feature types, and therefore different distributions of data per feature type in our experiments. For each of these real data sets, there are multiple different feature types.

Parameter Selection There were three data parameters in our experimental setup.

- Feature Type (FT): Number of feature types used to show the scalability of the algorithm.
- Cardinality of Feature Type (CF): Number of data points in each feature type.
- Cardinality of Queried Data (CQ): Number of data points in the queried data set.

Table 4.2 lists the characteristics of each data set and their parameter settings unless specified otherwise.

Data set	RAN	CLU	CA	NC
Dist	random	clustered	real	real
CF	2k to 10k	2k to 10k	4k	8k
CQ	20k to 100k	20k to 100k	22k	50k

Table 4.2: Data Set Description

Unless noted otherwise, we also chose the following parameters for the MTRNN algorithm based on empirical evaluation.

- R-Tree capacity (CR): The capacity of the R-Tree for each feature and queried data set was set to 36.
- Number of Subspaces (NS): The number of subspaces for generating feature routes was set to 30.

Experiment Design Figure 4.15 gives an overview of the experimental setup. The query processing engine takes the spatial data sets, the parameters to be applied on the data sets, and the baseline, the new 3-step MTRNN, and the classic RNN algorithms as input. The output consists of two categories of data, 1) performance measurements (execution time, number of IOs and filtering ratio) and 2) specific query results (RNNs). The performance measures are used to assess the viability of MTRNN for handling queries of various degrees of complexity. We include a comparison with the baseline algorithm, but this part of the evaluation is necessarily limited because baseline does not do any pruning of queried points, making it too time consuming to test more than a small number of feature types. We do not compare MTRNN with classic RNN on the performance measures listed above because the RNN algorithm is designed to solve classic RNN problems, not MTRNN problems. Since ours is the first formalization of the MTRNN problem, there are no other algorithms available to compare its performance with. Instead, we look to our second category of experimental output and assess the impact of MTRNN on the specific results returned compared to a traditional RNN approach. Note: we use RNNs or RNN points to refer to the query results of both MTRNN and classical RNN queries.

4.5.2 Evaluation Methodology

We evaluated the scalability of the MTRNN algorithm with the following questions:
(1) How do changes in number of feature types affect MTRNN performance?
(2) How do differences in cardinality for each feature type affect performance?
(3) How do differences in cardinality in the queried data set affect performance?
(4) How do changes in number of feature routes affect the filtering capability of the MTRNN algorithm?

We evaluated the impact of MTRNN on query results compared to classical RNN by asking:
(5) What is the percentage difference in number of RNNs returned for the two queries
(6) What is the percentage difference in specific RNN points returned?

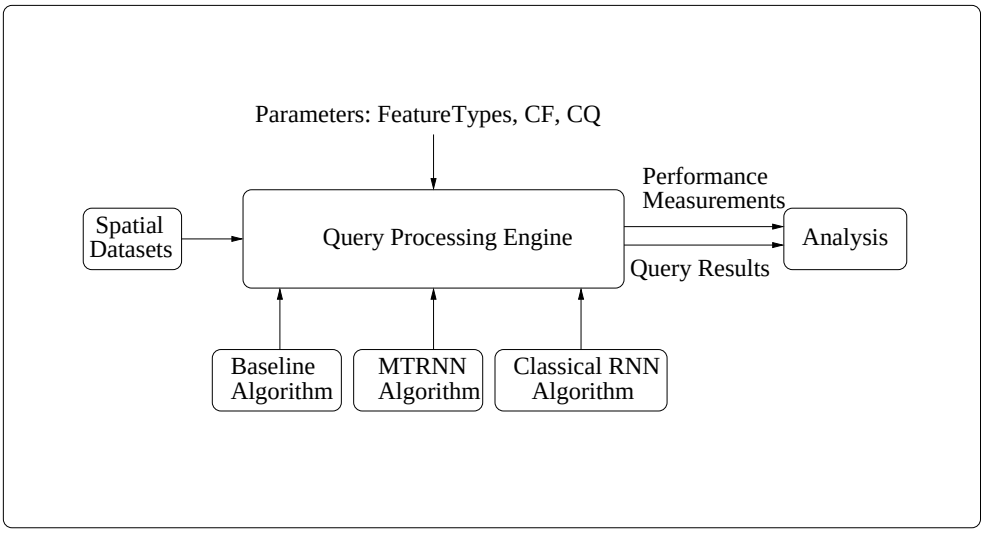


Figure 4.15. Experiment setup and design.

In every experiment for the MTRNN algorithm, we report CPU and IO time for the filtering and refinement steps, IO cost in terms of number of nodes accessed, and percentage of candidates remaining after filtering. We also report specific RNN points returned for both MTRNN and classical RNN algorithms. To reduce the effect of query point bias, we randomly selected 10 query points and averaged the results.

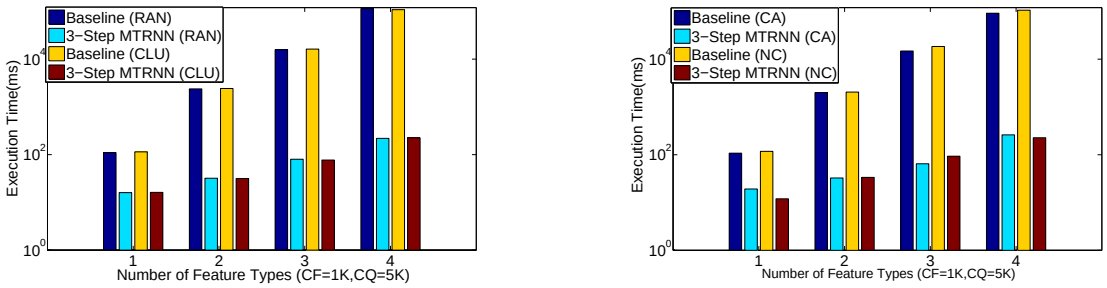
We define the following three metrics for evaluation purposes:
For evaluation of MTRNN algorithm performance:
(1) Filtering Ratio fr reflects the effectiveness of the filtering step.

$$fr = \frac{Number\ of\ filtered\ points\ before\ refinement}{NumPoints\ in\ queried\ data} \tag{4.7}$$

In order to directly show how the query results of MTRNN are different from RNN's, we define two metrics instead of use precision and recall.
For evaluation of the impact of MTRNN on query results compared to RNN:

(2)The percentage difference in number of RNN points p_n

$$p_n = \frac{\mid NumPoints\ in\ MTRNN - NumPoints\ in\ Classical\ RNN \mid}{NumPoints\ in\ Classical\ RNN} \tag{4.8}$$



(a) Synthetic (b) Real

Figure 4.16. Performance of baseline and MTRNN algorithms w.r.t. number of feature types.

(3)The percentage difference in specific RNN points p_s

$$p_s = \frac{NumPoints\ in\ MTRNN\ but\ not\ in\ ClassicalRNN}{NumPoints\ in\ Classical\ RNN} \tag{4.9}$$

In other words, p_n shows how much the new MTRNN query affects the answer to the question "how many RNNs are returned?". p_s indicates how significantly the new MTRNN query affects the answer to the question "what RNNs are returned?". p_n and p_s will be meaningless if no RNN is found for the classical RNN query.

4.5.3 Experimental Results

In this section, we present our evaluations of the MTRNN approach on both synthetic and real data. MTRNN performance evaluation includes runtime comparisons with the baseline algorithm using a small number of feature types followed by evaluations of MTRNN performance alone on various other measures. The final set of results presented measures the impact of the MTRNN approach on queries compared to traditional RNN. We end the section with a discussion of some important aspects.

Performance of Baseline and MTRNN Algorithms with regard to (w.r.t.) Number of Feature Types

We first evaluated the performance of the MTRNN algorithm against the baseline algorithm w.r.t. the number of feature types. Since the baseline algorithm is very time consuming, we only ran the experiments for $numFT$=1 to 4. We also chose a smaller subset of the data from both synthetic and real data sets with cardinality of feature data set 1k and cardinality of queried data set 5k form comparison purpose

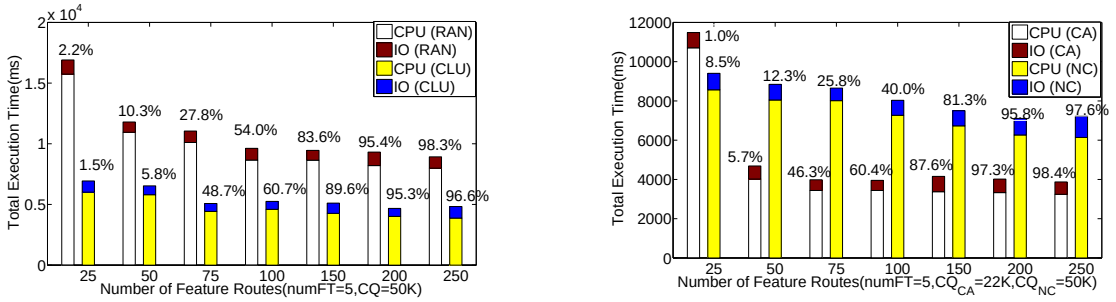
because the baseline algorithm for larger data set runs too long. As Figure 4.16 shows, the baseline execution time increases much more dramatically than for the 3-Step MTRNN algorithm as the number of feature types increases. On the other hand, our 3-Step MTRNN algorithm is much more scalable with the increment of feature types. When $numFT = 4$, The baseline execution time is more than two orders of magnitude longer. Similar patterns were found with increases in cardinality of feature type data and queried data.

Performance w.r.t. Number of Feature Routes

We then evaluated the performance of the MTRNN algorithm as the number of feature routes was raised from 25 to 250. As discussed in the previous section, the number of feature routes can be increased either by increasing the number of subspaces or mNN search in a subspace. As Figure 4.17 shows that both CPU and IO time decrease as the number of feature routes increases. However, when number of feature routes reaches a certain value (200 in the figure), performance gains are small, indicating that the filtering ratio is now increasing slowly. The reason is that pruning regions generated after 200 feature routes mostly overlap existing pruning regions. This result tells us that the number of feature routes in an MTRNN query can be set to a constant value. Our observation is supported by the cost model, where the effect of feature route number was set to be a constant value, too. This is true for both synthetic and real data sets.

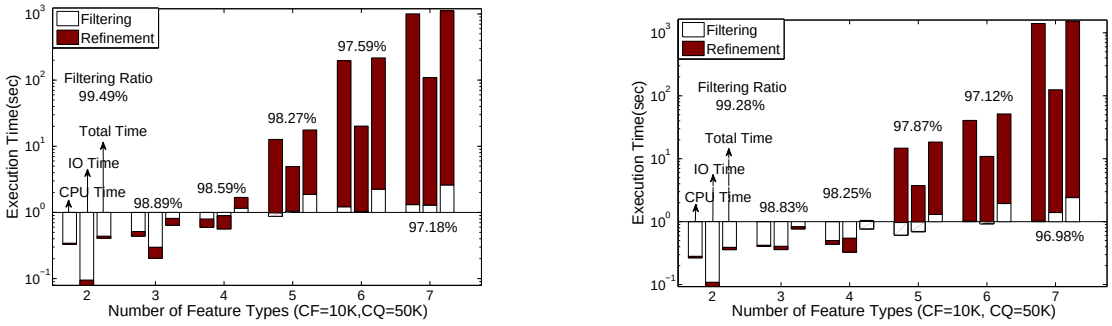
Scalability of MTRNN w.r.t. Number of Feature Types

Next we evaluated the scalability of the MTRNN algorithm with larger numbers of feature types than was tested in the first round of experiments. Overall, Figure 4.18



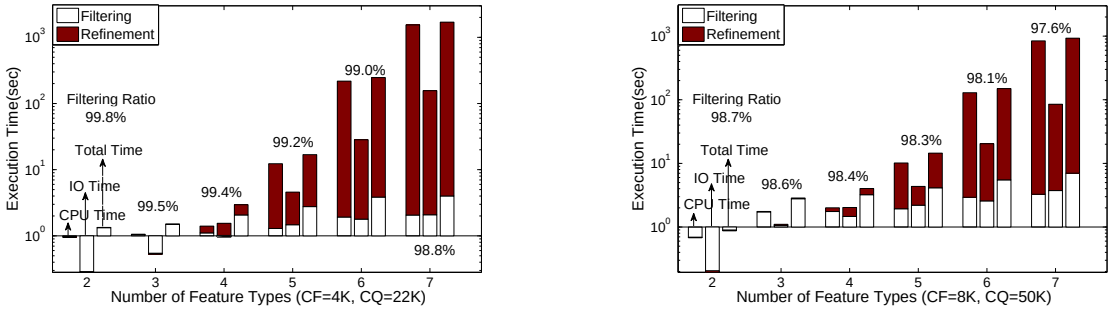
(a) Synthetic (b) Real

Figure 4.17. Performance w.r.t. number of feature routes.



(a) Random (b) Cluster

Figure 4.18. Scalability of MTRNN w.r.t. number of feature types on synthetic data sets.



(a) California (b) North Carolina

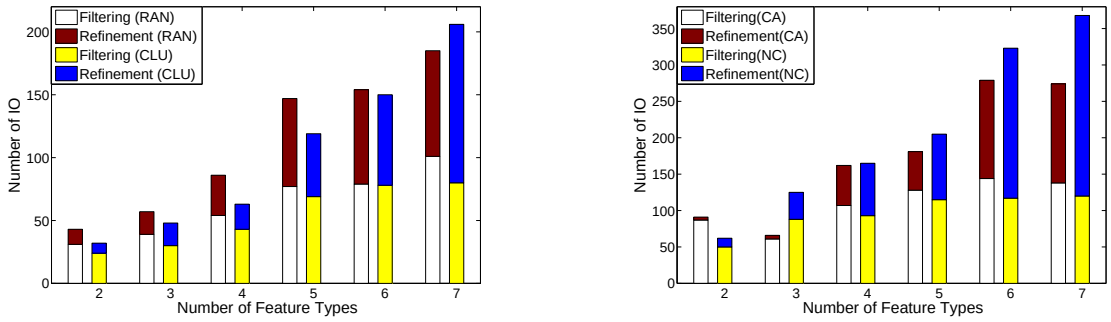
Figure 4.19. Scalability of MTRNN w.r.t. number of feature types on real data sets.

and Figure 4.19 show that the total time for processing up to 7 types, the maximum tested, is quite acceptable, considering the complexity of the MTRNN algorithm, the size of the data set and the experimental platform.

In addition, we can see the effect of feature type number on CPU and IO time. As shown in Figure 4.18a and 4.18b (in log scale) for the synthetic data sets, when the feature types number 4 or less, IO and CPU time have similar weights. When there are more than 4 feature types, CPU time and the refinement step become dominant ($FT > 4$). Meanwhile, the filtering ratio slightly decreases due to the longer feature routes and thus the looser distance bound.

The same trends are evident for the real data CA and NC in Figure 4.19a and 4.19b, again with CPU time and refinement step dominant for $FT > 4$. These results indicate that a simpler filtering algorithm works well enough for smaller numbers of feature types.

A closer look at the IO cost shown in Figure 4.20 reveals that while both filter-



(a) Synthetic Data (b) Real Data

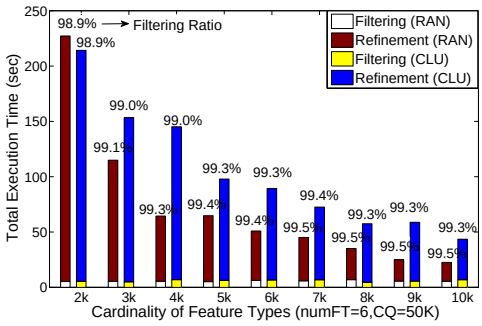
Figure 4.20. IO cost of MTRNN w.r.t. number of feature types.

ing and refinement IO costs grow when FT increases. refinement IO grows faster because the number of permutations involved in the final refinement step increases dramatically as FT increases.

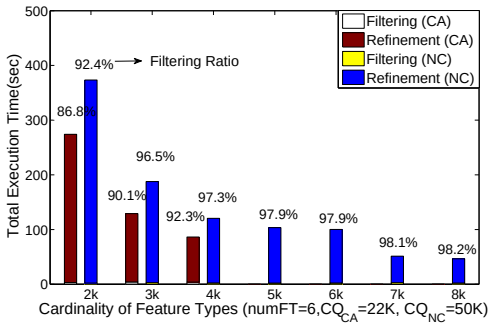
Scalability of MTRNN w.r.t. Cardinality of Feature Types

Next we looked at the scalability of the MTRNN algorithm w.r.t. cardinality of feature types. To better reflect the effects of cardinality, we used the same set of randomly selected query points to reduce the effect of biased queries. Since CPU time is dominant when FT=6, we only show total time in the figures. In Figure 4.21, we can see that for both synthetic and real data the run time of the MTRNN algorithm is less than 300 seconds for most cases, which is quite good for query problem of this complexity. Furthermore, run time decreases with increases in cardinality of FT. This is mainly due to the improved filtering ratio produced by the contribution of refined feature routes. These results indicate that our filtering algorithm works very well. (Since the maximum cardinality of the real CA data set is 4k, the experimental results from 5k to 8k are only for the real NC data.)

With similar settings, FT = 6 and changing the cardinality of feature types, Figure 4.22 shows the similar trend as Figure 4.21 for both synthetic and real data sets. In both cases, the queried data sets contain 100k points and we evaluated the performance for the 10k to 100k features data sets. The CA* and NC* data sets were generated with extra random data because the original data sets are not big enough. As the figures show, the run time decreases when increasing the cardinality of feature types. The maximum elapse time is about 150 seconds even with the large data sets, which indicates the algorithm scales very well.

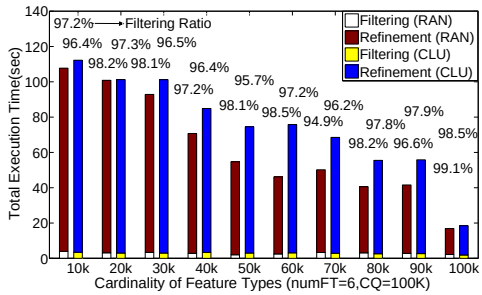


(a) Synthetic Data

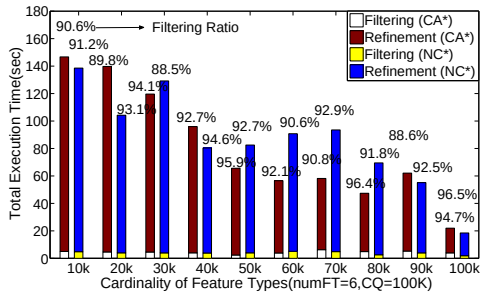


(b) Real Data

Figure 4.21. Scalability of MTRNN w.r.t. cardinality of feature types.



(a) Synthetic Data



(b) Real Data

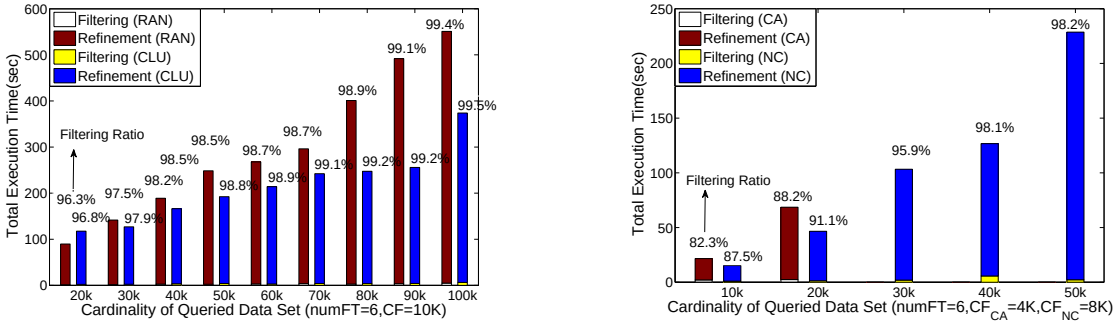
Figure 4.22. Scalability of MTRNN w.r.t. cardinality of feature types (Large Data Sets).

Scalability of MTRNN w.r.t. Cardinality of the Queried Data Set

We also examined the scalability of the MTRNN algorithm w.r.t. cardinality of the queried data (Figure 4.23). Contrary to the previous experiment, the trend this time is that increasing cardinality of the queried data increases run time (although no run exceeded 10 minutes even when the data set reached 100k).

Why does this happen? First, the filtering ratio is large enough and improves much less significantly than the CQ increments on this smaller range of space. Furthermore, because the cardinality of the data sets from the multiple feature types remains the same while the cardinality of the queried data set increases dramatically, the ratio of queried data to feature data also increases dramatically. Thus, more points are left as candidate MTRNNs after the filtering step, resulting in more MTRNN points found and a rise in total run time.

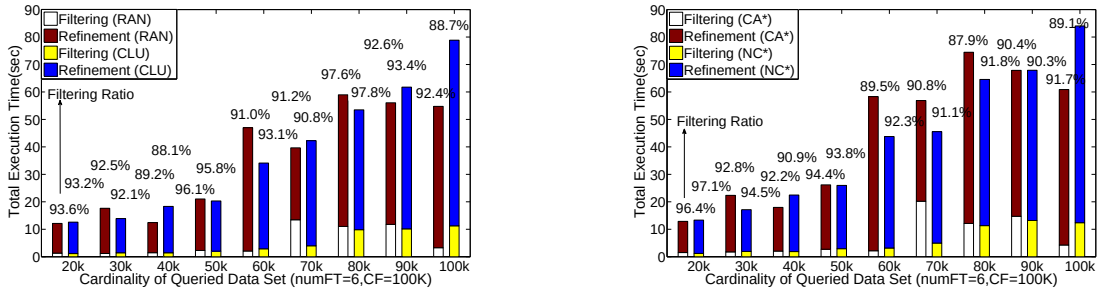
In Figure 4.24, the run time increases as the cardinality of queried data sets becomes larger. At this time, the size of queried data sets increase from 20k to 100k



(a) Synthetic Data (b) Real Data

Figure 4.23. Scalability of MTRNN w.r.t. Cardinality of the Queried Data Sets.

and the feature data set contains 100k points for both synthetic and real data sets, the run time changes from about 10 seconds to 90 seconds. As in section 4.5.3, the CA* and NC* data sets were generated with extra random data. This demonstrates that the algorithm can give the query results in a reasonable time range even with quite large queried data sets.



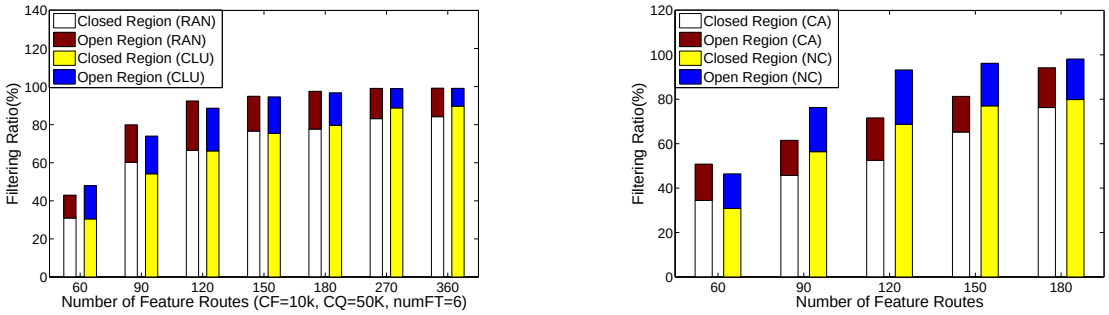
(a) Synthetic Data (b) Real Data

Figure 4.24. Scalability of MTRNN w.r.t. Cardinality of the Queried Data Sets (Large Data Sets).

Filtering Ratio of MTRNN w.r.t Number of Feature Routes

Next we evaluated how the number of feature routes affects the filtering ratio. As shown in Figure 4.25, the filtering ratio is very significant when the number of feature routes exceeds 150, which indicates our filtering algorithm is quite efficient.

Figure 4.25 shows the filtering ability for both closed and open region pruning methods. Since closed regions are pruned before open regions, more points are pruned by closed region pruning. As shown in Figure 4.25a for synthetic data, the filtering



(a) Synthetic Data

(b) Real Data

Figure 4.25. Filtering ratio of MTRNN w.r.t. number of feature routes.

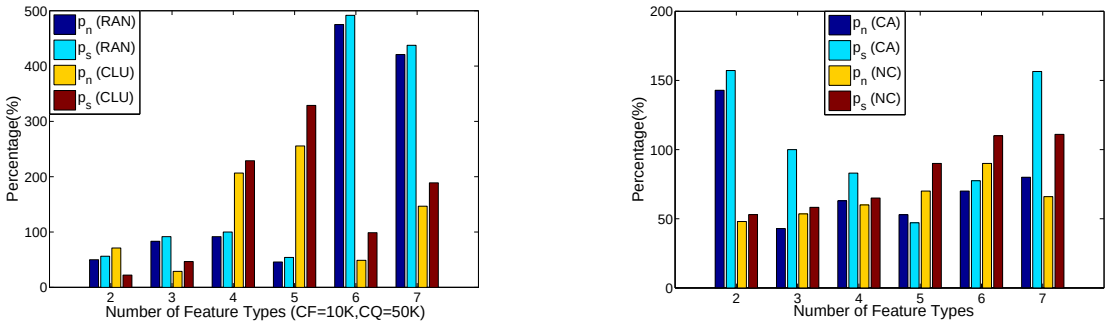
ratio grows as we increase the number of feature routes. However, when feature routes reach a certain number (180 in the figure), the growth of the filtering ratio becomes much less significant. Therefore, to save running time, we limited the number of space divisions to 30 in our experiments (resulting in about 180 feature routes). Figure 4.25a also indicates that both closed and open region pruning contributes to efficiency of the filtering step. The filtering ratio for a closed region pruning is generally higher because it is performed before open region pruning.

Figure 4.25b for real data exhibits a similar pattern. That is, when the number of feature routes reaches a certain value, the filtering ratio is more than 90% and becomes steady.

Since we were finding one nearest neighbor in one subspace in this experiment, the results also show how filtering efficiency improves with increasing number of subspaces.

Change in Number and Specific RNN Points Returned for MTRNN and classical RNN Queries w.r.t. Number of Feature Types

Finally, to highlight the potential impact of the MTRNN query approach, we quantified changes in number of RNNs and specific RNNs w.r.t. FT for MTRNN and classical RNN queries. The results illustrated in Figure 4.26 show that p_n and p_s are indeed significant for both synthetic and real data. For synthetic data (Figure 4.26a), as the FT increases, the smallest percentage change of p_n with FT = 3 for data set CLU is more than 20% and most of the percentages reached more than 50%. The figure also shows that the MTRNN algorithm tends to find more RNNs when FT is relatively large.



(a) Synthetic Data (b) Real Data

Figure 4.26. Change of RNNs w.r.t. number of feature types.

The differences are even more dramatic in RNNs returned for the real datasets Figure 4.26b, where the value of p_n and p_s rarely falls below 50% even when the number of features is only 2 or 3. In other words, MTRNN queries consistently generated results that differed by more than 50% from those generated by classical RNN queries.

Discussion

Factors that Impact MTRNN Algorithm Performance The performance of the MTRNN algorithm is usually significantly affected by the filtering ratio for large data sets. The refinement step of the MTRNN algorithm is usually dominant especially with larger feature type number (> 4 in our experiment), because the MTNN algorithm applied during this step is very time consuming with feature type number rising. The larger the filtering ratio, the less the MTNN algorithm is called during the refinement step (one MTNN is called for one candidate MTRNN). Therefore, improvement of overall performance depends largely on how many candidates can be pruned during the filtering step. As can be seen from Figure 4.21 and 4.23, the filtering ratio improves as the range of the data increases (cardinality of features and queried data). The reason is that faraway R-Tree nodes are more likely to be pruned. Since the filtering step is less dominant in terms of execution time with large feature type number, we may also increase the number of feature routes to achieve similar effects. Nevertheless, a good filtering algorithm will always help to increase filtering ability.

On the other hand, when number of feature types is small, total execution time is not significant. In this case IO time becomes dominant and performance could be

improved by increasing the number of feature routes (thus filtering ratio) and cache size to store more R-Tree nodes in memory.

Impact of Number of MTRNN Query on Results Summarizing the results shown in Figure 4.26, it is clear that MTRNN queries can generate dramatically different results from classical RNN queries, both in terms of quantity (How many RNNs are there?) and content (What are the RNNs?). In general, the overlap between the results of MTRNN and RNN queries decreases as the feature type number increases because the other feature types around the queried data set will have an impact, too. Finally, it is important to understand that p_n and p_s could be quite different for the same set of data, with major implications for RNN-based decision-making. For example, in Figure 4.26 for the CA data set with three feature types p_n is very small because the number of MTRNN results does not change much. However, the actual results returned are totally different from the classical RNN query results, resulting in a very high value of p_s. Deciding whether to look at p_n, p_s or both depends entirely on the application.

4.6 Summary

We formalized a multi-type reverse nearest neighbor problem (MTRNN) and developed an MTRNN query algorithm by exploiting a two-step R-tree node-level pruning strategy. In the coarse-level pruning step, we generate closed and open pruning regions which can be used to prune an entire R-tree or part of an R-tree for a given query. In the refinement step, the remaining data points are evaluated and the true MTRNN points are identified for the given query. We compared the MTRNN algorithm with a traditional RNN query method in terms of number of feature types, number of points in each feature type and queried data set. The experiment results show that our algorithm not only returns MTRNNs within a reasonable time but that MTRNN results differ significantly from results of traditional RNN queries. This finding has important implications for decision-making algorithms used in business and confirms the value of further investigation of MTRNN query approaches.

As for future work, we plan to introduce different weight factors for different feature types into the objective function. It is important for some business applications since in reality the influence of different feature types may be different for different types of applications. We are then interested in the application of MTRNN queries

on road networks using road network distance or other types of spatio-temporal databases. We believe that the computational complexity of MTRNN queries will rise dramatically when applied to such databases. Therefore, we plan to explore off-line techniques for pre-computing of intermediate results of MTRNN queries as well as indexing, data structure, and other methods for efficient storage and retrieval of results. Another direction to extend our MTRNN work is to design some good heuristic approaches to find approximate results as well as design corresponding measurements to evaluate the results that sufficiently capture the problem's complexity.

Chapter 5

Conclusions and Future Work

Spatial database research confronts daunting challenges in today's computing environment. The growing complexity of spatial database and GIS systems has been accompanied by rising user expectations of spatial databases, GIS systems and their applications. Users are looking to location-based services to handle all manner of reasonable traveler queries. Therefore, it becomes increasingly critical to expand search capabilities in spatial databases and GIS-related applications.

In this thesis we extended the scope of classic nearest neighbor (NN) and reverse nearest neighbor (RNN) search problems by incorporating multiple feature types into their formulations and devising new algorithms to solve them. We began by considering the case of a traveler who visits locations that belong to different categories, or feature types during a single trip, such as a post office, a gas station, and a grocery store. Traditional NN query methods are insufficient in this basic traveling scenario because they assume only one or two feature types. What needs to be found is the shortest route going through one instance from every feature type. We capture this notion by formalizing a multi-type nearest neighbor (MTNN) problem [40], which answers a query that is different from traditional NN search. The MTNN query is a valuable addition to location-based services in spatial databases and geospatial applications.

However, it is not enough as we did in [40] to use Euclidean distance as the measurement to find the shortest route in MTNN problems because a majority of queries in real life are to find a route in real road networks. Therefore, we formalized another

search problem called BEst Start Time Multi-Type Nearest Neighbor (BESTMTNN) problem [38] that gives a turn-by-turn route and best start time in terms of travel time.

Similar to traditional NN search problems, reverse nearest neighbor search problems traditionally consider the influence of an instance of a single feature type on the queried points. Thus, they cannot answer queries regarding the influence of more than one feature type, such as a grocery store and a wine shop. For the cases where travelers may visit multiple instances of more than one feature type, however, the influence of multiple feature types instead of one feature type must be considered, which requires extension to the scope of the classic RNN query. Multi-Type Reverse Nearest Neighbor (MTRNN) problem [41] was formalized to find the influence of multiple feature types on the queried points.

5.1 Major Results

Specifically the primary contributions of this thesis are the following:

- We formalized and studied a generalized MTNN query problem. We pointed out the difference between the MTNN problem and the Traveling Salesman Problem (TSP) [48], which is one of the class of "NP Complete" combinatorial problems. We designed an algorithm based on our page-level upper bound (PLUB) pruning technique at the R-tree node level in order to filter a large block of spatial data in the early stages. This algorithm found the optimal solution to approach the MTNN problem when the feature type number is small. We also provided an algebraic cost model for PLUB and compared it to the RLORD algorithm. Our experiments showed that our method outperforms RLORD with clustered data sets although the optimal solution becomes computationally intractable when the number of query feature types is large.

- We identified the special properties of a new type of query involving spatial and temporal features on spatio-temoral road networks. Then we formalized the BESTMTNN query problem on spatio-temporal road networks by extending the encoded path view from spatial-only to spatio-temporal road networks and designed a label-correcting based algorithm that grew the spatio-temporal partial route as the time window was extended. This algorithm prioritized the

spatio-temporal partial routes with current least travel time. The input to the algorithm is a user-specified query involving spatio-temporal features such as query time window sizes for all features and planned stay-time interval at a location. The output is a turn-by-turn route and the best start time in terms of least travel time. Our experiments showed our algorithm could answer normal user BESTMTNN queries in a reasonable time.

- We illustrated the motivation of the MTRNN problem with examples to show how current RNN query techniques may give misleading answers in some applications. We formalized the MTRNN query problem to consider the influence of other feature types in addition to the feature type of the given query point. Based on the concepts of MTRNN query and the algorithm of MTNN query, we gave a brute force algorithm as a base line algorithm. R-tree based algorithms were then proposed to prune the search space with our new pruning techniques called closed region pruning and open region pruning. These two pruning techniques filtered R-tree nodes and points that were definitely not the MTRNN of the given query point. Then we applied three refinement approaches to remove the false hit points. Our experiments on both synthetic and real data sets showed our algorithms run much faster than the baseline algorithm. The experiment also demonstrated that typical MTRNN queries could be answered by our algorithms within reasonable time for different experimental settings. The design decisions related to performance tuning were provided as well. To highlight the significance of the MTRNN query, we also included experimental results that vividly show the degree to which MTRNN query results could differ from results of traditional RNN queries.

5.2 future Research Directions

This thesis identifies a new area in spatial database research that is of fundamental importance to travel-related GIS applications. The work here expands and enriches the potential of location-based services. There are several directions to extend the current work.

The first direction is to incorporate continuous query techniques into the MTNN search. A continuous MTNN search can give more precise answers according to changing traffic conditions. As shown in the thesis, MTNN searches are very complicated.

Designing proper data structures and storage methods for continuous MTNN queries is a daunting challenge. Existing frameworks for continuous queries would require careful adaptation in order to avoid compromising the efficiency of the MTNN query algorithm. In addition, MTNN algorithms themselves would need to be redesigned for the continuous query environment.

The BESTMTNN query and algorithm are an attempt to apply the MTNN query in the real world. However, the work has two limitations. First, the Time Aggregated Multi- Type Graph (TAMTG) is a single layer model of the road networks. Therefore, the performance of the BESTMTNN algorithm will degrade dramatically when the road networks become very large. One way to extend the BESTMTNN work on road networks is to design a hierarchical spatio-temporal graph to model such networks as well as new algorithms based on these models. The second limitation of the BESTMTNN work is that only long term-traffic information was incorporated into the TAMTG. Within modern computing systems such as mobile services, real time traffic information is needed to update the MTNN query results from time to time when a traveler is on the road. This requires enhancing the TAMTG and designing related new algorithms for it.

As with MTNN queries, MTRNN queries also need to be made applicable to road networks. Because the computational complexity of MTRNN queries rises dramatically when applied to spatio-temporal databases, it is worth exploring off-line techniques for pre-computing of intermediate results of MTRNN queries as well as indexing, data structure, and other methods for efficient storage and retrieval of results.

Another way to extend MTRNN work is to incorporate the notion of influence of "multiple features" into different application domains. Evaluating the influence of multiple feature types likely requires different approaches in different domains. How to model the "accumulated" influence of multiple features for different real world settings depends on the context of the domains and requires deep thinking and understanding. In other words, adapting MTRNN queries for different domains cannot be done without specific knowledge of the domains. Formalizing problems similar to MTRNN in different domains is challenging and designing good algorithms to prune the search space is even harder.

Considering the complexity of MTNN and MTRNN queries, another interesting direction to extend the current work is to investigate heuristic algorithms to find

approximate results as well as design corresponding measurements to evaluate the results that sufficiently capture the problems complexity. A user-defined tolerance factor can be used to control the quality of the query results found by heuristic algorithms. Here, the challenge will be to design proper tolerance factors for different environments without losing efficiency of the algorithms.

To conclude, this thesis presented the author's attempt to extend traditional NN and RNN queries in order to meet the growing demands of complex computation environments. In the future, more extensions to the current work are needed to make MTNN and MTRNN queries viable in real world applications.

Bibliography

[1] North Carolina Center for Geographic Information and Analysis, NC Master Address Dataset Project. http://www.cgia.state.nc.us/Services/NCMasterAddress, 2009.

[2] GPS Data Team, GPS POI Files. http://poidirectory.com/poifiles/, 2010.

[3] E. Achtert, C. Bahm, P. Krager, P. Kunath, A. Pryakhin, and M. Renz. Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces. In *ACM SIGMOD*, 2006.

[4] R. Benetis, C. Jensen, G. Karciauskas, and S. Saltenis. Spatio-Temporal Network Databases and Routing Algorithms: a Summary of Results. In *IDEAS*, 2002.

[5] S. Berchtold, C. Bohm, D. Keim, and H. Kriegel. A Cost Model for Nearest Neighbor in High-Dimensional Data Space. 1997.

[6] S. Berchtold, C. Bohm, D. Keim, and H. Kriegel. On Optimizing Nearest Neighbor Queries in High-Dimensional Data Spaces. 2001.

[7] L. Bodin and B. Golden. Classification in vehicle routing and scheduling. *Networks*, 1981.

[8] H. Chen, W. Ku, M. Sun, and R. Zimmermann. The Multi-Rule Partial Sequenced Route Query. In *ACM GIS*, 2008.

[9] B. Cherkassky, A. Goldberg, and T. Tadzik. Shortest paths algorithm: theory and experimental evaluation. *Mathematical Proggramming*, 1996.

[10] K. Cheung and A. Fu. Enhanced Nearest Neighbor Search on the R-tree. *ACM SIGMOD Record*, pages 16–21, 1998.

[11] K. Clarkson. Fast Algorithms for the All-Nearest-Neighbors Problem. In *FOCS*, 1983.

[12] J. G. Cleary. Analysis of an algorithm for finding nearest neighbor in Euclidean space. *ACM Transactions on Mathematical Software*, pages 183–192, June 1979.

[13] A. Corral, Y. Manolopoulos, and M. Vassilakopoulos. Closest Pair Queries in Spatial Databases. In *ACM SIGMOD*, 2000.

[14] A. Corral, Y. Manolopoulos, and M. Vassilakopoulos. Algorithms for Processing K-closest-pair Queries in Spatial Databases. *Data and Knowledge Engineering*, pages 67–104, 2004.

[15] A. Corral, M. Vassilakopoulos, and Y. Manolopoulos. The impact of Buffering on Closest Pairs Queries Using R-Trees. In *Proceedings of Advances in Databases and Information Systems (ADBIS'01)*, pages 41–54, 2001.

[16] Y. Cui, B. C. Ooi, K. Tan, and H. V. Jagadish. Indexing the Distance: An Efficient Method to KNN Processing. In *VLDB*, 2001.

[17] C.Yang and K.-I. Lin. An Index Structure for Efficient Reverse Nearest Neighbor Queries. In *ICDE*, 2001.

[18] E. Dellis and B. Seeger. Efficient Computation of Reverse Skyline Queries. In *VLDB*, 2007.

[19] B. Ding, J. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *EDBT*, 2008.

[20] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logrithmic expected time. *ACM Transactions on Mathematical Software*, pages 209–226, September 1977.

[21] Y. Gao, B. Zheng, G. Chen, W.-C. Lee, K. Lee, and Q. Li. Visible Reverse k-Nearest Neighbor Queries. In *ICDE*, 2009.

[22] B. George, S. Kim, and S. Shekhar. Spatio-temporal network databases and routing algotihms: a summary of results. In *SSTD*, 2007.

[23] H. Herhatosmanoglu, D. A. I. Stanoi, and A. Abbadi. Constrained Nearest Nieghbor Queries. In *SSTD*, 2001.

[24] C. Hjaltason and H. Samet. Incremental Distance Join Algorithms for Spatial Databases. In *ACM SIGMOD*, 1998.

[25] C. Hjaltason and H. Samet. Distance Browsing for Spatial Databases. *ACM Transactions on Database Systems*, pages 265–318, 2 1999.

[26] Y.-W. Huang, N. Jing, and E. Rundensteiner. A semi-materialized view approach for route maintenance in Intelligent Vehicle Highway systems. In *ACM GIS*, 1994.

[27] Y.-W. Huang, N. Jing, and E. Rundensteiner. Hierarchical encoded path view for path query processing: an optimal model and its performance evaluation. *IEEE TKDE*, pages 409–432, May/June 1998.

[28] J. M. Kang, M. Mokbel, S. Shekhar, T. Xia, and D. Zhang. Continuous Evaluation of Monochromatic and Bichromatic Reverse Nearest Neighbors. In *ICDE*, 2007.

[29] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *ACM SIGMOD*, 1997.

[30] F. Korn, F. A. Com, S. Muthukrishnan, and D. Srivastava. Reverse Nearest Neighbor Aggregates Over Data Streams. In *ACM SIGMOD*, 2001.

[31] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *ACM SIGMOD*, 2000.

[32] F. Korn, S. Muthukrishnan, and D. Srivastava. Reverse nearest neighbor aggregates over data stream. In *VLDB*, 2002.

[33] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 1992.

[34] G. Laporte, M. Gendreau, J. Potvin, and F. Semet. Classical and modern heuristics for the vehicle routing problem. *Intl. Transactions in Operational Research*, 2000.

[35] R. Levin, Y. Kanza, E. Safra, and Y. Sagiv. Interactive Route Search in the Presence of Order Constraints. In *VLDB*, 2010.

[36] F. Li, D. Chen, M. Hadjieleftherious, G. Kollios, and S. Teng. On trip planning queries in spatial databases. In *SSTD*, 2005.

[37] X. Lian and L. Chen. Monochromatic and Bichromatic Reverse Skyline Search over Uncertain Databases. In *ACM SIGMOD*, 2008.

[38] X. Ma, S. Shekhar, and H. Xiong. Multi-Type Nearest Neighbor Queries on Road Networks with Time Window Constraints. In *ACM SIG SPATIAL GIS*, 2009.

[39] X. Ma, S. Shekhar, H. Xiong, and P. Zhang. Exploiting a page-level upper bound for multi-type nearest neighbor queries. In *Technique Report,05-008, University of Minnesota*, 2005.

[40] X. Ma, S. Shekhar, H. Xiong, and P. Zhang. Exploiting Page Level Upper Bound for Multi-Type Nearest Neighbor Queries. In *ACM GIS*, 2006.

[41] X. Ma, C. Zhang, S. Shekhar, Y. Huang, and H. Xiong. On Multi-Type Reverse Nearest Neighbor Search. In *Journal Of Data and Knowledge Engineering DOI 10.1016/j.datak.2011.06.003*, 2011.

[42] Muhammad Cheema and Xuemin Lin and Wei Wang and Wenjie Zhang and Jian Pei. Probabilistic reverse nearest neighbor queries on uncertain data. In *TKDE*, 2010.

[43] Muhammad Cheema and Xuemin Lin and Ying Zhang and Wei Wang and Wenjie Zhang . Lazy updates: An efficient technique to continuously monitoring reverse knn. In *VLDB*, 2009.

[44] D. Papadias, Y. T. Q. Shen, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, 2004.

[45] A. Papadopoulos and Y. Manolopoulos. Performance of Nearest Neighbor Queries in R-trees. In *ICDT*, pages 394–408, 1997.

[46] H. peter Kriegel, P. Kroger, M. Renz, A. Zufle, and A. Katzdobler. Reverse k-Nearest Neighbor Search Based on Aggregate Point Access Methods. In *SSDBM*, 2009.

[47] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, 1985.

[48] G. Reinelt. *The Traveling Salesman - Computational Solutions for TSP Applications*. Springer Verlag, 1994.

[49] N. Roussopoulos and F. V. S. Kelly. Nearest Neighbor Queries. In *SIGMOD*, 1995.

[50] M. Safar, D. Ebrahimi, and D. Taniar. Voronoi-Based Reverse Nearest Neighbour Query Processing on Spatial Networks. In *ACM Multimedia Systems Journal (MMSJ), Volume 15, Issue 5, pp. 295-308, published by Springer*, 2009.

[51] T. Seidl and H. Kriegel. Optimal Multi-Step k-Nearest Neighbor Search. In *SIGMOD*, 1998.

[52] J. Shan, D. Zhang, and B. Salzberg. On Spatial-Range Closest-Pair Query. In *SSTD*, 2003.

[53] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. In *University of Southern California, Computer Science Department, Technical Report 05-840*, January 2005.

[54] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The Optimal Sequenced Route Query. In *VLDB Journal DOI 10.1007/s0078-006-0038-6*, 2007.

[55] M. Sharifzadeh and C. Shahabi. Processing optimal sequenced route queries using voronoi diagrams. In *Geoinformatica DOI 10.1007/s10707-007-0034-z*, 2008.

[56] A. Singh, H. Ferhatosmanoglu, and A. S. Tosun. High Dimensional Reverse Nearest Neighbor Queries. In *CIKM*, 2003.

[57] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *SSTD*, 2002.

[58] I. Stanoi, D. Agrawal, and A. E. Abbadi. Reverse Nearest Neighbor Queries for Dynamic Databases. In *SIGMOD workshop on Research Issues in data mining and knowledge discovery*, 2000.

[59] I. Stanoi, I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Discovery of Influence Sets in Frequently Updated Databases. In *VLDB*, 2001.

[60] Y. Tao, D. Papadias, and X. Lian. Reverse kNN Search in Arbitrary Dimensionality. In *VLDB*, 2004.

[61] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *VLDB*, 2002.

[62] Y. Tao, M. L. Yiu, and N. Mamoulis. Reverse Nearest Neighbor Search in Metric Spaces. In *TKDE, 7(4-5)*, 2006.

[63] Y. Tao, J. Zhang, D. Papadias, and N. Mamoulis. An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces. In *TKDE*, 2004.

[64] Q. T. Tran, D. Taniar, and M. Safar. Bichromatic Reverse Nearest-Neighbor Search in Mobile Systems. In *IEEE Systems Journal, Vol. 4, Issue 2, pp. 230-242*, 2010.

[65] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Norvag. Reverse Top-k Queries. In *ICDE*, 2010.

[66] R. C.-W. Wong, T. Ozsu, P. Yu, and A. W. Fu. Efficient Method for Maximizing Bichromatic Reverse Nearest Neighbor . In *VLDB*, 2009.

[67] W. Wu, F. Yang, C.-Y. Chan, and K.-L. Tan. Continuous Reverse k-Nearest-Neighbor Monitoring. In *MDM*, 2008.

[68] W. Wu, F. Yang, C.-Y. Chan, and K.-L. Tan. Finch: Evaluating Reverse k-Nearest-Neighbor Queries on Location Data. In *VLDB*, 2008.

[69] T. Xia and D. Zhang. Continuous Reverse Nearest Neighbor Monitoring. In *ICDE*, 2006.

[70] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On Computing Top-t Most Influential Spatial Sites. In *VLDB*, 2005.

[71] C. Yang and K.-I. Lin. An Index Structure for Improving Nearest Closest Pairs and Related Join Queries in Spatial Databases. In *IDEAS*, 2002.

[72] B. Yao, F. Li, and P. Kumar. Reverse Furthest Neighbors in Spatial Databases. In *ICDE*, 2009.

[73] M. L. Yiu and N. Mamoulis. Reverse Nearest Neighbors Search in Ad-hoc Subspaces. In *ICDE*, 2006.

[74] M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao. Reverse Nearest Neighbors Search in Large Graphs. In *TKDE,18(4):540-553*, 2006.

[75] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao. All-Nearest-Neighbors Queries in Spatial Databases. In *SSDBM*, 2004.